OBJECTWINDOWS™ FOR C++

REFERENCE GUIDE



BORLAND

ソフトウェアのご使用条件

この度は、Borland International 社（以降ボーランド）の製品をお買い上げいただき誠にありがとうございます。

このソフトウェアは、ボーランドの規定した『ナンセンスでないライセンス規定』およびボーランドジャパンの規定した『カスタマーサポート』に同意いただいた場合のみ提供させていただくことになっています。ご熟読のうえ、条件に同意された場合のみご開封ください。下記の規定をご熟読のうえ、添付されている「ボーランドジャパンカスタマーサポートセンター登録書」をボーランドジャパンあてにご返送ください。登録書をご返送いただいた方は、弊社でユーザー登録をおこないバージョンアップのご案内などを行います。登録書は、楷書体で丁寧にご記入ください。

登録書をご返送いただけない場合でも、フロッピーディスクの包装を解いた時点で、下記の規定にご同意いただけたものといたします。下記の規定にご同意いただけない場合は、ご購入されたパッケージを購入伝票などを添えて返品して下さい(ただし返送料金はお客様負担とさせていただきます)。フロッピーディスクの包装を解いた製品は、返品できません。

これらのライセンス規定および保証に関する質問がある場合は、カスタマー・サポート・センター［電話：03-5489-6671（言語製品)／03-5489-6991（データベース製品)］までご連絡下さい。

ナンセンスでないライセンス規定および制限付保証

1） 著作権

本ソフトウェアは米国著作権法、国際著作権法によって保護されています。

2） コピーの制限

本製品に含まれる、マニュアルの一部または全部を許諾なく複製することを禁じます。ただし、本製品のプログラムディスクを、使用するコンピュータに組み込むためのコピー、および破損または紛失などの損害から守るための保存コピー（バックアップ・コピー）のみ許諾します。

3） 使用権

「書籍」と同様にお考え下さい。1冊の書籍を2つの場所で、2人が同時に読むことはできません。それと同様に、本ソフトウェアを2つの場所で、2人が同時に使用することはできません。ただし、本ソフトウェアを同時でなければ複数の人が利用でき、複数のコンピュータの間を自由に移動することが可能です。（もちろん、ボーランドの著作権を侵害しないという条件付です。）

同時に複数の人が、複数の場所でご使用の場合は、人数分の製品の購入が必要です。

4） コンパイルしたプログラムの使用

ボーランド言語コンパイラを使って書き、コンパイルしたプログラムは「著作権通知」を行った場合に限り新たなライセンスを取得したり、ライセンス料金を支払うことなく使用、寄贈または販売をすることが可能です。「著作権通知」とは、作成者本人の著作権を表示するか、ディスクラベルに記載されているボ

ーランド言語コンパイラの著作権通知のいずれかを意味します。
ボーランド言語コンパイラには、ランタイムライブラリが使用する字体情報とコード化ハードウェアをはじめとする種々のサポートファイルが含まれます。
ボーランド言語コンパイラを使って作成したプログラムを、使用、寄贈または販売する場合にも「著作権表示」をして下さい。さらに、ボーランド言語コンパイラを使って書き、コンパイルしたプログラムがこれらのサポートファイルを利用している場合は、このサポートファイルを上記プログラムと組み合わせてプログラムの全てのコピーに「著作権通知」をして使用、配布及び販売をして下さい。このサポートファイルを別個に使用、寄贈または販売することを禁止します。

5) **サンプルプログラムとツールボックスルーチンの使用**
本ソフトウェアに含まれる全てのデモ用のサンプルプログラムは、教育的目的にのみ使用が限定されます。これらのサンプルプログラムを編集または修正したり、自分で使用するために書いたプログラムに組み込むことは許可しますが、それらを、実行可能形式オブジェクト、またはソースコード形式の、単独プログラムまたは、そのプログラムの一部として、寄贈、販売、使用許可を与えたり、別の形式で配布することは認めません。
本製品サンプルプログラムルーチンとツールボックス・ルーチンは、作成されたプログラム全体に占める割合が小さく、外観、機能、設計がサンプルプログラムとほとんど類似性が無い場合、ルーチンをプログラムに組み込むことを認めます。本製品のサンプル・プログラムやツールボックスからのルーチンを含むプログラムは、実行可能な書式においてのみ、無制限に、ライセンス料を支払うことなく配布することができます。もちろん、本製品のルーチンを含む皆様自身の製品から発生したすべての賠償請求、責任おび損害については、皆様自身が単独でその責任を負い、それから発生する被害からボーランドを守るものとします。

6) **再販、譲渡などの禁止**
本製品の一部または全部を譲渡、賃貸販売することは認めません。

制限付保証

ボーランドは、同封するディスケット本体とドキュメンテーションが、購入後60日間は物理的不備や欠陥がないことを保証します。保証期間中に、材料または仕上げに欠陥が生じたことを書面を以て通知を受けると、ボーランドは、この通知が正しいことを確認し、欠陥ディスケットまたはドキュメンテーションを交換します。この制限付保証の違反に対する責任および救済処置は、欠陥ディスケットまたはドキュメンテーションの交換に限定され、ボーランドがこのような損害が生じる可能性について予め知らされている場合であっても、いかなる損害賠償の責任も負いません。損害には、これだけに限定されるものではありませんが、利益やデータの損失、ソース・プログラムの使用、特殊な損害、偶発的または必然的損害、またはその他の類似の請求が含まれます。いかなる場合においても、皆様またはその他の人々が受けた損害に対するボーランドの責任は、その賠償請求形式を問わず、ソース・プログラムを使用するライセンス料の表示価格または実際に支払われた価格を超えないものとします。

ボーランドは、明示的または黙示的保証を含め、他のいかなる保証も拒否します。黙示的保証には、これだけに限定されるものではありませんが、市場性または具体的目的への適合性が含まれます。具体的には、ボーランドは、このソース・プログラムが特定の目的に適合することを主張または保証しておらず、市場性に対する黙示的保証は、ディスケットおよびドキュメンテーションのみを対象とした(ソース・プログラムは含まれません)限定保証の保証期間60日に限定され、それ以外については、明白かつ明瞭に拒否されます。

準拠法と一般規定

本ライセンスおよび制限付保証は、カリフォルニア州法に準拠して解釈、規制されるものとします。本パッケージ中のコンピュータ・ソフトウェアおよびドキュメンテーションを米国政府が使用、複写または公開する場合も、市販のコンピュータ・ソフトウェアにて起用される非公開権（DFARS 52227-7013）に準拠するものとします。本文書に明記されていない全ての権利は、ボーランドが保留します。

『カスタマーサポート』

「ボーランドジャパンカスタマーサポートセンター登録書」に記名、捺印のうえ、弊社までご返送いただいた場合に限り、「ボーランドジャパンカスタマーサポートセンター」のサービスが受けられます。ただし、このサービスは本ソフトウェア購入時より１年間とさせていただきます。また、このサービスは登録カードにご記入されたご使用者１名様のみが受けられます。

1）郵便、電話、FAXによるお問い合わせ

ご連絡先　〒150　東京都渋谷区恵比寿西2-6-11　川上ビル
㈱ボーランドジャパン
カスタマーサポートセンター
電話番号　03-5489-6671（言語製品）
03-5489-6991（データベース製品）
ファクシミリ　03-5489-4873
営業時間　祝祭日を除く月～金曜
10:00～12:00、13:00～17:00

お問い合わせの際は、かならずシリアルナンバー（マスターディスクの１枚目に記載されています）をご連絡ください。ご連絡いただけない場合は、お答えいたしかねます。また、プログラミング技法に関するご質問にはお答えできません。

2）マスターディスクの再コピー（有料）

なんらかの原因でマスターディスクを破損された場合、再コピーサービスを受けることができます。破損されたディスクと再コピー実費3,000円（税別）を、弊社業務課までお送りください。

3) **仕様の変更**

本ソフトウェアの仕様は、改良のため予告なく変更されることがあります。あらかじめご了承ください。

ボーランドジャパンカスタマーサポートセンター登録書ご記入方法

ご記入いただきました登録書にもとづき、アップデート、新製品情報、その他のご案内をご登録住所に送付いたします。

ご記入は明確にわかるように、楷書体でお願いします。「フリガナ」がある項目については必ずご記入ください。

1) **登録書のご記入について**

■すでに弊社ユーザID番号をお持ちの方

登録内容に変更が無ければ、氏名（法人･団体の場合は担当者氏名）、お買い上げ日、お買い上げ店名のみのご記入で結構です。

登録内容に変更がある場合、変更項目に□を付けて変更内容をご記入ください。

■ ID番号をお持ちでない方

すべての項目にご記入ください（ユーザ登録書を弊社にご送付後、ユーザ登録完了のご通知とともにユーザID番号をお知らせします）。

①個人で登録の方

住所、電話番号は必ず自宅住所、自宅電話番号を記入して下さい。

法人・団体名欄には何も記入しないで下さい。

②法人・団体（会社、学校等）で登録の方

氏名欄には、担当者氏名をご記入ください。

■ ID番号について

ID番号は、シリアル番号、MSA-NET(BBS)のID番号とは異なりますのでご注意ください。

2) **登録内容の変更について**

登録された内容（住所、電話番号、所属、担当者等）に変更が生じた場合、すみやかに書面（ハガキ、手紙、FAX）で必ずお知らせください。お知らせいただきませんと各種ご案内が送付できません。

ObjectWindows™ for C++

リファレンスガイド

Borland International
1800 Green Hills Road
P.O. Box 660001
Scotts Valley, CA 95067-0001

編訳　ボーランド株式会社 1991

目　次

第3章　ObjectWindowsのその他の構成要素 ……147

第2部
WindowsAPI クイックリファレンス ……169

第4章　Windows 関数 ……171

第7章　Windows 定数 ……………641

はじめに

このマニュアルは2部構成になっており，第1部はObjectWindowsのアプリケーションフレームワークに関する詳細リファレンス，第2部はMicrosoft WindowsのAPI（アプリケーションプログラムインターフェース）に関するクイックリファレンスです。

ObjectWindowsを使うとWindowsアプリケーションの開発が簡単になります。ObjectWindowsの定義済みクラスには，Windowsメッセージを自動的に処理する汎用のアプリケーションクラスをはじめ，ウィンドウ，ダイアログ，およびリストボックスやボタンなどのコントロール，その他のユーザインターフェース要素を簡単に作成するためのクラスが数多く用意されています。なお，チュートリアル，プログラム例，プログラミングの説明については，**ObjectWindows for C++プログラマーズガイド**を参照してください。

第1部は以下の3章に分かれます。

■第1章『**ObjectWindows クラス**』

ObjectWindowsの主階層構造に属するクラスの一覧です。それぞれのデータメンバ，メンバ関数，演算子，フレンド，その他の関連関数を解説します。

■第2章『**ストリームクラス**』

ライブラリにストリーム機能を実現するクラスの一覧です。各クラスのメンバ，演算子，フレンドを解説します。

■第3章『**ObjectWindows のその他の構成要素**』

ObjectWindowsで定義する型，定数，変数，マクロ，クラス，関数の一覧です。ObjectWindows要素のうち階層内のクラスでないものをまとめてあります。

Windows APIのフレームワークには，関数，マクロ，メッセージ，データ構造体，データ型，スタイル，定数があります。Borland C++で開発しながら，Windows関数をすべて利用できます。ただし，ObjectWindowsのクラスで充分用が足りれば使う必要はありません。Windows APIについては，**ObjectWindows for C++プログラマーズガイド**で詳しく解説しています。他に，**プログラミングWINDOWS Version 3（アスキー出版局）**も参照してください。

第2部は，以下の4章に分かれます。

- 第4章『**Windows関数**』

  Windows APIを構成する関数とマクロのアルファベット順一覧です。
- 第5章『**Windowsメッセージ**』

  WindowsとWindows環境で実行されるアプリケーションとの対話に使われるメッセージの一覧です。
- 第6章『**Windows型**』

  単純な型や構造体などWindowsで定義されている型を解説します。
- 第7章『**Windows定数**』

  Windowsで定義済みの共通の定数とスタイルを分類し，解説します。この章の定数は，ObjectWindowsのメンバ関数呼び出しで引数として使われます。

命名規約と書体

このマニュアルで使う命名規約と書体は，**ObjectWindows for C++プログラマーズガイド**の『**はじめに**』で説明しています。

その他のObjectWindows命名規約

ObjectWindowsでは以下の関連型も使います。

- **PC*char***：char定数へのポインタ。
- **P*char***：char型へのポインタ。
- **PC*void***：void定数へのポインタ。

- **P***int*：int 型へのポインタ。
- **P***void*：void 型へのポインタ
- **RC***Object*：*Object* 定数への参照。
- **R***int*：整数型への参照。
- **R***ipstream*：**ipstream** 型への参照。
- **R***opstream*：**opstream** 型への参照。
- **RP***void*：void 型を指すポインタへの参照。

P A R T 1

第1部

ObjectWindows リファレンス

第1章

ObjectWindows クラス

この章では，ObjectWindows のすべての標準クラスをアルファベット順に記載し，使用法，メンバ，演算子，フレンドを解説します。

クラスの調べ方

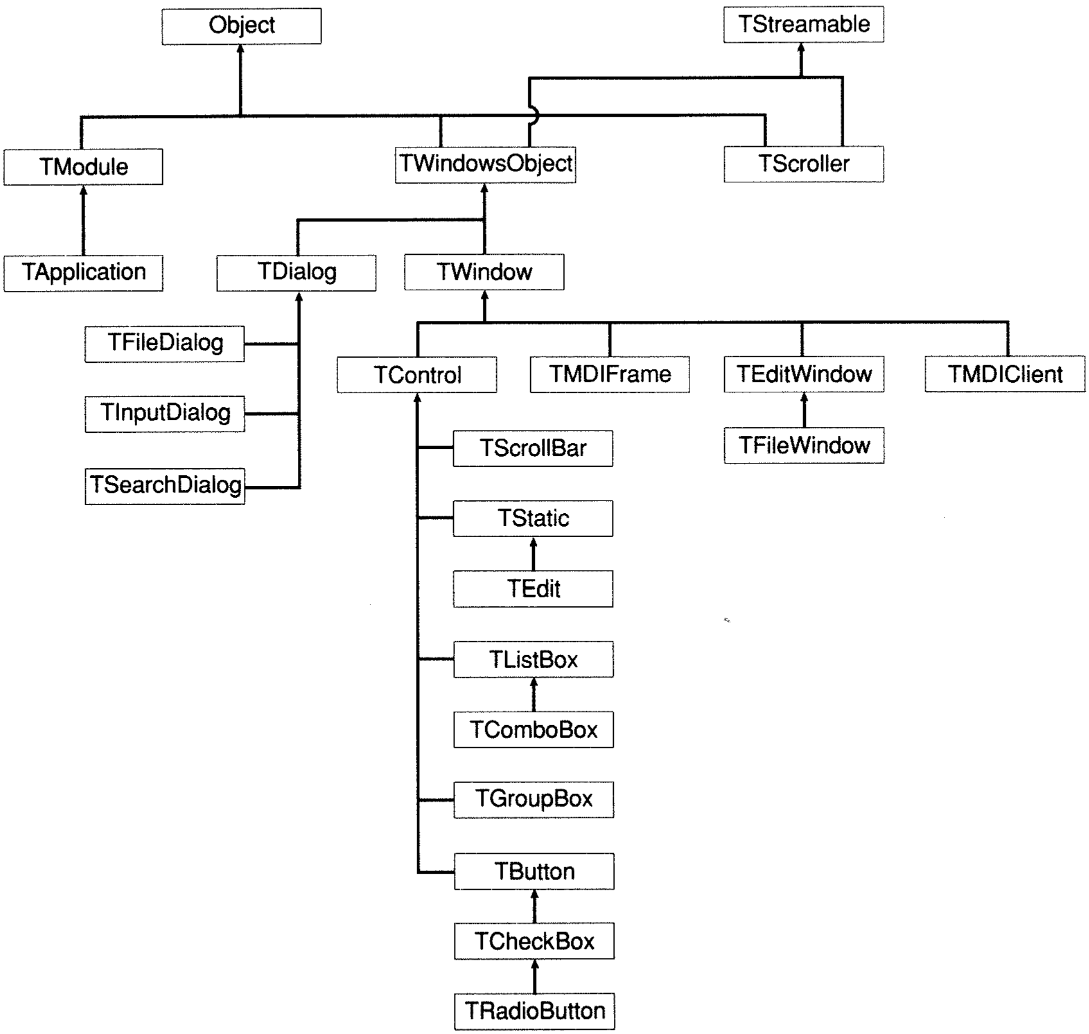
階層を構築している各クラスは，基本クラスから多くの特性を継承しているということをまず念頭に置いてください。この章では，クラスのデータメンバとメンバ関数のうち，そのまま継承されたもの以外，すなわち新しいものと変更があったものだけを記載しています。

継承したメンバについては索引で探してください。もちろん，階層をさかのぼって探すこともできます。

たとえば，***TEdit*** クラスのデータメンバ ***Parent*** を例にとります。まず，***TEdit*** のデータメンバを探しても ***Parent*** の記載はないはずです。そこで ***TEdit*** の階層から ***TStatic*** というすぐ上の基本クラスを探します。そこにも記載はありません。さらに ***TStatic*** のすぐ上にある ***TControl*** という基本クラスを調べます。このように階層をさかのぼると ***TWindowsObject*** に行き着きます。そこに ***TEdit*** が継承している ***Parent*** が詳しく解説されています。**図 1.1** でこの階層を確かめてください。

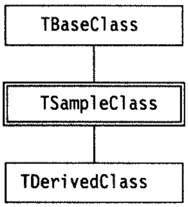
この章は，**図 1.1** に示すクラスのリファレンスです。***TStreamable*** については，第2章**『ストリームクラス』**を参照してください。

図 1.1
ObjectWindows のクラス階層



各クラスの見出しと説明は次の形式になっています。

TSampleClassName クラス　サンプルクラスのヘッダファイル



クラスの概要，他クラスとの関係，一般的な使用法を述べ，さらに続けてデータメンバ，メンバ関数，関連関数を列記します。

ObjectWindows の各型や ***EM_OUTOFMEMORY*** などの定数については第 3 章『**ObjectWindows のその他の構成要素**』に詳しく載っています。また，***LPSTR*** などの Windows の各型については，第 6 章『**Windows 型**』を参照してください。

アクセス制限のあるプロテクトメンバ関数には，プログラムテキスト（宣言）の右に protected と書いてあります。アクセス制限がないパブリック関数には何も書いてありません。なお，プライベートメンバは記載してありません。

データメンバ

各クラスに新しく登場するデータメンバをアルファベット順に記載し，各メンバの宣言，使用制限（プロテクトメンバの時），使用法を示します。

ADataMember　SomeType ADataMember ;

ADataMember は，このサンプルクラスのデータメンバです。このテキストはデータメンバの内容と使用法を示します。

関連項目：関連するデータメンバ，メンバ関数，クラス，グローバル関数，その他。

AnotherDataMember　AnotherType AnotherDataMember ;　　protected

上の *ADataMember* と同様です。

メンバ関数

このクラスで新しく定義されたメンバ関数，または，継承したメンバ関数を定義し直したメンバ関数をすべて記載します。

Constructor TSampleClassName (SomeType AParameter) ;

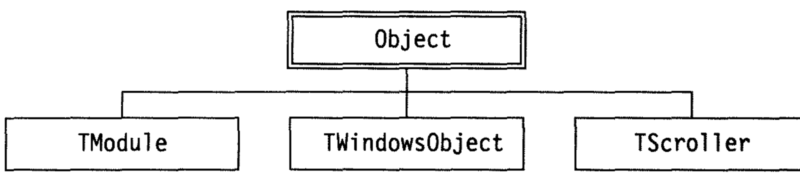
新しいサンプルクラスのコンストラクタです。データメンバ *ADataMember* に *AParameter* を設定します。

Zilch virtual void Zilch () ;

サンプルクラスに特定の処理を実行させます。

関連項目： ***TSomethingElse* : : *Zilch***

Object OBJECT.H



ObjectWindows 階層のルートです。派生オブジェクトの役割を定義する抽象クラスで，型チェックとカプセル化のための基本的な機構と構造を規定します。

データメンバ

ZERO static PObject ZERO ;

プライベートクラスが使うエラーオブジェクトのポインタです。演算子 new がオブジェクトのスペースを確保できない場合の問題を処理します。

メンバ関数

constructor Object () ;

Object のインスタンスを構築します。

constructor Object (RObject) ;

コピーコンストラクタ。データメンバを *Object* の別のインスタンスにコピーします。

destructor virtual ~Object () ;

仮想デストラクタです。*Object* のインスタンスを破棄します。

firstThat virtual RCObject firstThat (condFuncType, Pvoid) const ;

指定されたテスト関数を呼び出して，オブジェクトと引数リストを探させます。条件を満たす最初のオブジェクトをコンテナから探します。テストが成功すればオブジェクトを，失敗すれば *NOOBJECT* を返します。

forEach virtual void forEach (iterFuncType, Pvoid) ;

コンテナ内の各オブジェクトに対して，指定された繰り返し処理を引数リストつきで呼び出します。

hashValue virtual hashValueType hashValue () const = 0 ;

派生クラスの純粋仮想関数です。オブジェクト別の固有キーであるハッシュ値を返します。

isA virtual classType isA () const = 0 ;

派生クラスの純粋仮想関数です。クラス固有のクラス識別子を返します。

isAssociation virtual int isAssociation () const ;

オブジェクトが *Association* の派生クラスであるかどうかを判定します。基本クラス *Object* の場合は 0 を返します。

isEqual virtual int isEqual (RCObject) const = 0 ;

派生クラスの純粋仮想関数です。引数のオブジェクトと等しいかどうかをテストします。

isSortable virtual int isSortable () const ;

オブジェクトがソート可能かどうか，つまり ***Sortable*** の派生クラスであるかどうかを判定します。ソートできない基本オブジェクト ***Object*** の場合は 0 を返します。

lastThat virtual RCObject lastThat (condFuncType, Pvoid) const ;

非コンテナオブジェクトの場合は ***FirstThat*** と同様です。コンテナオブジェクトの場合は定義し直され，指定の条件関数が 1 を返す最後のオブジェクトへの参照を返します。

nameOf virtual Pchar nameOf () const = 0 ;

派生クラスの純粋仮想関数です。クラスを識別する文字列を返します。

new Pvoid operator new (size_t) ;

オブジェクトに指定のバイト数を確保します。確保できなければ，ゼロを返します。

printOn virtual void printOn (Rostream) const = 0 ;

派生クラスの純粋仮想関数です。オブジェクトのプリントアウト表現をストリームに書き込みます。

フレンド

operator<< Rostream operator << (Rostream, RCObject) ;

出力ストリームにオブジェクト値を書き込む ***Object*** のフレンド関数です。

関連関数

operator==　int operator == (RCObject test 1, RCObject test 2) ;

最初のオブジェクトと次のオブジェクトが同じかどうかを判定します。

**operator != **　int operator != (RCObject test 1, RCObject test 2) ;

最初のオブジェクトと次のオブジェクトが異なるかどうかを判定します。

operator＞＞と operator＜＜

以下のストリームクラスはすべて4つの演算子を関連関数として宣言します。

TButton, TCheckBox, TComboBox, TDialog, TEdit, TEditWindow, TFileDialog, TFileWindow, TInputDialog, TGroupBox, TListBox, TMDIClient, TMDIFrame, TRadioButton, TScrollBar, TScroller, TStatic, TWindow, TWindowsObject

このマニュアルでは，以上のクラスをまとめて ***TClassName*** という名前を使います。

operator＞＞と operator＜＜がそれぞれ2つずつあります。多重継承による曖昧さを解消するためだけ使います。

2つの operator＞＞関数のうち，1つ目は ***TClassName*** オブジェクトへの**参照**をとり，2つ目は ***TClassName*** オブジェクトを指す**ポインタの参照**をとります。同じように，2つの operator＜＜関数のうち，1つ目は ***TClassName*** オブジェクトへの**参照**をとり，2つ目は ***TClassName*** オブジェクトを指す**ポインタの参照**をとります。

operator>> Ripstream operator >> (Ripstream is, RTClassName cl) ;
Ripstream operator >> (Ripstream is, RPTClassName cl) ;

入力ストリーム *is* から ***TClassName*** オブジェクトを読み込み，それを *cl* に書き込みます。ストリームへの参照が返され，＞＞演算子の通常の連結が行なえます。

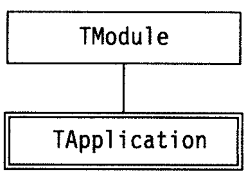
関連項目：***ipstream***

operator<< Ropstream operator << (Ropstream os, RTClassName cl) ;
Ropstream operator << (Ropstream os, PTClassName cl) ;

TClassName オブジェクト *cl* を出力ストリーム *os* に書き込みます。ストリームへの参照が返され，＜＜演算子の通常の連結が行なえます。

関連項目：***opstream***

TApplication APPLICAT.H



TModule から派生し，Windows アプリケーションモジュールの「代役」として動作します。***TApplication*** と ***TModule*** は Windows アプリケーションに必須の基本的な動作を提供します。***TApplication*** のメンバ関数はメインウィンドウの作成など，インスタンスの初期化とメッセージ処理を行ないます。

データメンバ

HAccTable HANDLE HAccTable ;

アプリケーションで使う現在の Windows アクセラレータテーブルのハンドル

を保持します。

hPrevInstance HANDLE hPrevInstance ;

前に実行した Windows アプリケーションのインスタンスのハンドルが入ります。いま実行中のインスタンス以前に実行されたインスタンスがなければ 0 です。

KBHandlerWnd PWindowsObject KBHandlerWnd ;

キーボードハンドラ機構が有効であれば，現在のアクティブウィンドウを指すポインタです。ObjectWindows ではこのメンバを使って，コントロールつきのウィンドウにキーボード入力を処理させます。このウィンドウでは，タブまたは矢印キーが押されるとコントロールのフォーカスが別のコントロールに移動します。

MainWindow PTWindowsObject MainWindow ;

アプリケーションのメインウィンドウを指すポインタです。

nCmdShow int nCmdShow ;

メインウィンドウを通常どおりに表示するか，アイコンで表示するかを示す整数です。値については，Windows 関数 ***ShowWindow*** を参照してください。ObjectWindows では，アプリケーションのメインウィンドウを表示するのに使います。

メンバ関数

constructor TApplication (LPSTR AName, HANDLE AnInstance,
HANDLE APrevInstance, LPSTR ACmdLine,
int ACmdShow) ;

TApplication オブジェクトを構築し，そのデータメンバを指定された引数で初期化します。***TModule*** のコンストラクタを呼び出します。

destructor ~TApplication () ;

TApplication のオブジェクトを破棄します。***MainWindow*** が残っていれば，削除します。

CanClose virtual BOOL CanClose () ;

アプリケーションをクローズしてもよければ真を返します。デフォルトではメインウィンドウのメンバ関数 ***CanClose*** を呼び出し，その戻り値を返します。このメンバ関数はめったに定義し直しません。クローズ時の動作を定義し直す場合はメインウィンドウのメンバ関数 ***CanClose*** で行ないます。

関連項目：***TWindowsObject*** : : ***CanClose, TWindowsObject*** : : ***WMDestroy***

idleAction virtual void IdleAction () ; protected

アプリケーションがアイドル状態のときのメッセージループ中に呼び出されます。どのアプリケーションの Windows メッセージキューにもメッセージがないときに処理を行ないます。「無動作（"do-nothing"）」メンバ関数であり，派生クラスで定義し直せば「アイドル状態」での処理を指定できます。

InitApplication virtual void InitApplication () ; protected

アプリケーションで最初に実行されるインスタンスにだけ必要な初期化を行ないます。デフォルトでは何もしません。派生クラスで定義し直すと，アプリケーション固有の初期化を行なえます。

InitInstance virtual void InitInstance () ; protected

アプリケーションで実行される各インスタンスに必要な初期化を行ないます。***InitMainWindow*** を呼び出し，さらに ***TModule*** : : ***MakeWindow*** と ***TWindowsObject*** : : ***Show*** を呼び出してメインウィンドウ要素の作成と表示を行なわせます。メインウィンドウを作成できなかった場合，データメンバ ***Status*** に ***EM_INVALIDMAINWINDOW*** がセットされます。このメンバ関数を定義し直す場合は ***TApplication*** : : ***InitInstance*** を明示的に呼び出してください。

関連項目：***TApplication*** : : ***InitMainWindow,***
TApplicaton : : ***InitApplication, TApplication*** : : ***Run,***
TModule : : ***MakeWindow, TWindowsObject*** : : ***Show***

InitMainWindow virtual void InitMainWindow () ; protected

デフォルトでは，アプリケーション名をタイトルに持つ汎用クラス ***TWindow*** のオブジェクトを構築します。***InitMainWindow*** を定義し直して，派生クラス

で使える実用的なメインウィンドウオブジェクトを構築し，***MainWindow*** に格納してください。一般的な使用法を以下に示します。

```
virtual void TMyApp : : InitMainWindow ()  {
    MainWindow  =  new TMyWindow (NULL, " Caption") ;
}
```

isA virtual classType isA () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TApplication*** のクラス識別子である ***applicationClass*** を返します。

MessageLoop virtual void MessageLoop () ; protected

アプリケーションの存続中ずっと実行されるメッセージループを操作します。メッセージの有無を Windows に問い合わせ，メッセージを受け取ると ***ProcessAppMsg*** を呼び出して処理します。問い合わせに対してメッセージがなければ，***IdleAction*** を呼び出してアイドル時間の処理を行なわせます。

関連項目：***TApplication*** : : ***ProcessAppMsg, TApplication*** : : ***IdleAction***

nameOf virtual Pchar nameOf () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TApplication*** のクラス識別文字列である「TApplication」を返します。

ProcessAccels virtual BOOL ProcessAccels (LPMSG PMessage) ; protected

アクセラレータ専用のメッセージを処理します。アプリケーションのウィンドウをアクセラレータに応答させない場合は，このメンバ関数をオーバーライドして偽を返すだけにしたほうが効率的です。

関連項目：***TApplication*** : : ***ProcessAppMsg***

ProcessAppMsg virtual BOOL ProcessAppMsg (LPMSG PMessage) ; protected

モードなしダイアログボックス，アクセラレータ，MDI アクセラレータからのメッセージに対する専用の処理をチェックします。***ProcessDlgMsg***, ***ProcessMDIAccels***, ***ProcessAccels*** を呼び出し，専用メッセージが 1 つでもあれば真を返します。アプリケーションがモードなしダイアログボックスを作成せず，アクセラレータに応答せず，MDI アプリケーションでもなければ，こ

のメンバ関数をオーバーライドして偽を返すだけにしたほうが効率的です。

関連項目：***TApplication*** :: ***ProcessDlgMsg***, ***TApplication*** :: ***ProcessAccels***, ***TApplication*** :: ***ProcessMDIAccels***

ProcessDlgMsg　virtual BOOL ProcessDlgMsg (LPMSG PMessage) ;　protected

コントロールへのキーボード入力を処理するため，モードなしダイアログとモードなしウィンドウからのメッセージを処理します。アプリケーションのモードなしダイアログボックスとモードなしウィンドウにコントロールをつけない場合は，このメンバ関数をオーバーライドして偽を返すだけにしたほうが効率的です。

関連項目：***TAppication*** :: ***ProcessAppMsg***

ProcessMDIAccels　virtual BOOL ProcessMDIAccels(LPMSG PMessage) ;　protected

MDI 対応アプリケーション専用のアクセラレータメッセージを処理します。MDI に準拠しないアプリケーションでは，このメンバ関数をオーバーライドして偽を返すだけにしたほうが効率的です。

関連項目：***TApplication*** :: ***ProcessAppMsg***

Run　virtual void Run () ;

インスタンスを初期化します。他に実行中のインスタンスがなければ，***InitAppliation*** を呼び出します。その後で ***InitInstance*** を呼び出します。初期化が成功してデータメンバ ***Status*** が 0 になれば，***MessageLoop*** を呼び出し，アプリケーションを起動します。

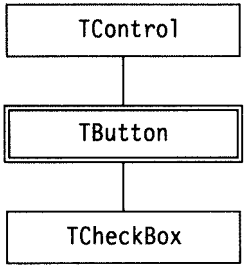
関連項目：***TApplication*** :: ***InitApplication***, ***TApplication*** :: ***InitInstance***, ***TApplication*** :: ***MessageLoop***

SetKBHandler　void SetKBHandler (PTWindowsObject AWindowsObject) ;

KBHandlerWnd に ***AWindowsObject*** をセットして，指定されたウィンドウに対するキーボード処理をアクティブにします。これにより，キーボード入力はコントロール選択項目に変換されます。なお，このメンバ関数は ObjectWindows によって内部的に呼び出されます。

関連項目：***TWindowsObject*** :: ***EnableKBHandler***, ***TWindowsObject*** :: ***WMActivate***

TButton



Windows のプッシュボタンインターフェース要素を表わすインターフェースクラスです。親の ***TWindow*** でボタンコントロールを作成するときは ***TButton*** を使わなければなりません。また ***TButton*** を使うと，アプリケーションと ***TDialog*** のボタンコントロール間の通信が簡単になります。

プッシュボタンは 2 種類あります。細枠で表示される通常のボタンと，太枠で表示されるデフォルトボタンです。デフォルトボタンは**リターン**キーが押された場合のウィンドウのデフォルト動作を表わします。1 つのウィンドウに置けるデフォルトボタンは 1 つだけです。

データメンバ

IsDefPB BOOL IsDefPB ;

デフォルトのプッシュボタンかどうかを示します。オーナー描画ボタンに使われ，***BS_DEFPUSHBUTTON*** スタイルの設定に基づきリソースでない ***TButton*** のコンストラクタによって設定されます。

メンバ関数

constructor TButton (PTWindowsObject AParent, int AnId, LPSTR AText, int X, int Y, int W, int H, BOOL IsDefault, PTModule AModule = NULL);

渡された親ウィンドウ *AParent*，コントロールID*AnId*，対応テキスト *AText*，親ウィンドウのクライアント領域の原点を基準とする座標 *X*, *Y*，幅 *W*，高さ *H* を使ってボタンオブジェクトを構築します。また，同様の引数をとる ***TControl*** のコンストラクタを呼び出します。*IsDefault* が真であれば，***Attr.Style*** に設定されている ***TButton*** のデフォルトスタイルに ***BS_DEFPUSHBUTTON*** を追加します。真でない場合は ***BS_PUSHBUTTON*** を追加します。

関連項目：***TControl*** :: ***TControl***

constructor TButton (PTWindowObject AParent, int ResourceId, PTModule AModule = NULL);

TDialog のボタンコントロールに対応する ***TButton*** オブジェクトを構築します。同じ引数をとる ***TControl*** のコンストラクタを呼び出します。転送するデータがないので，さらに ***DisableTransfer*** を呼び出して，転送機構からボタンを除外します。

引数 *ResourceId* は，ユーザが定義するボタンリソースに必ず対応させてください。

関連項目：***TControl*** :: ***TControl, TWindowsObject*** :: ***DisableTransfer***

constructor TButton (StreamableInit); **protected**

TButton のストリームコンストラクタです。***TControl*** のストリームコンストラクタを呼び出します。入力ストリームからのデータを使って ***TButton*** のインスタンスを構築するときに呼び出されます。

BMSetStyle virtual void BMSetStyle (RTMessage Msg) = [WM_FIRST + BM_SETSTYLE] ;

オーナー描画ボタンの場合，Windows が ***BS_DEFPUSHBUTTON*** スタイルを設定しようとすると，オーナー描画ボタンがデフォルトのプッシュボタンか

どうかを確かめます。オーナー描画ボタン以外の場合は，単に ***DefWndProc*** を呼び出します。

build static PTStreamable build () ;

TButton (***StreamableInit***) コンストラクタを呼び出します。***TButton*** 型のオブジェクトを構築した後で，そのデータメンバがストリームから読み込まれます。

GetClassName virtual LPSTR GetClassName () ; protected

TButton の Windows 登録クラス名「BUTTON」を返します。

関連項目：***TWindowsObject*** : : ***GetWindowClass***

SetupWindow virtual void SetupWindow () ; protected

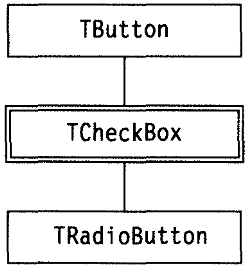
ボタンがデフォルトのプッシュボタンであり，オーナー描画ボタンでもある場合，***DM_SETDEFID*** メッセージを親ウィンドウに送ります。

WMGetDlgCode virtual void WMGetDlgCode (RTMessage Msg) = [WM_FIRST + WM_GETDLGCODE] ;

ダイアログマネージャからの ***WM_GETDLGCODE*** メッセージに応答します。オーナー描画ボタンの場合，ボタンがデフォルトボタンかどうかの情報を返します。オーナー描画ボタン以外の場合は，単に ***DefWndProc*** を呼び出します。

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。



Windowsのチェックボックスインターフェース要素を表わすインターフェースオブジェクトです。親の ***TWindow*** でチェックボックスコントロールを作成するときは ***TCheckBox*** を使わなければなりません。また，***TCheckBox*** を使うとアプリケーションと ***TDialog*** のチェックボックスコントロール間の通信が簡単になります。

チェックボックスには**チェックされている状態**と**チェックされていない状態**の2つの状態があります。**無効（淡色）**状態を加えた3状態チェックボックスもあります。***TCheckBox*** のメンバ関数は主にチェックボックスの状態を管理します。コントロールを形と機能別にまとめたグループ（***TGroupBox***）に，チェックボックスを所属させることもできます。

データメンバ

Group PTGroupBox Group

当該チェックボックスを他のチェックボックスと，またはラジオボタン（***TRadioButton***）と論理的にグループ化する ***TGroupBox*** コントロールオブジェクトのポインタです。チェックボックスが属するグループがなければ，NULLになります。

関連項目：***TGroupBox, TRadioButton***

メンバ関数

constructor TCheckBox (PTWindowsObject AParent, int AnId, LPSTR ATitle, int X, int Y, int W, int H, PTGroupBox AGroup, PTModule AModule = NULL) ;

渡された親ウィンドウ（*AParent*），コントロールID（*AnId*），対応テキスト（*ATitle*），親ウィンドウのクライアント領域の原点を基準とする座標（*X*, *Y*），幅（*W*），高さ（*H*），対応グループボックス（*AGroup*）を使ってチェックボックスオブジェクトを構築します。***TCheckbox*** :: ***TCheckbox*** がチェックボックスのデータメンバ ***Attr.Style*** に ***WS_CHILD*** | ***WS_VISIBLE*** | ***WS_TABSTOP*** | ***BS_AUTOCHECKBOX*** をセットします。***TButton*** のコンストラクタを呼び出します。

関連項目：***TButton*** :: ***TButton***

constructor TCheckBox (PTWindowsObject AParent, int ResourceId, PTGroupBox AGroup, PTModule AModule = NULL) ;

TDialog のチェックボックスコントロールに対応する ***TCheckBox*** オブジェクトを構築します。*ResouceId* は必ずリソースファイルのチェックボックスリソースに対応させてください。***TButton*** のコンストラクタを呼び出します。***Group*** に *AGroup* をセットします。次に ***EnableTransfer*** を呼び出して転送機構を有効にします。

関連項目：***TButton*** :: ***TButton, TWindowsObject*** :: ***EnableTransfer***

constructor TCheckBox (StreamableInit) ; protected

TCheckBox のストリームコンストラクタです。***TButton*** のストリームコンストラクタを呼び出します。入力ストリームからのデータメンバを使って ***TCheckBox*** のインスタンスを構築するときに呼び出されます。

関連項目：***TButton*** :: ***TButton***

BNClicked virtual void BNClicked (RTMessage Msg)
= [NF_FIRST + BN_CLICKED] ; protected

チェックボックスがクリックされたことを知らせる通知メッセージに自動的に応答します。チェックボックスの ***Group*** が NULL でなければ，***TGroupBox*** のメンバ関数 ***SelectionChanged*** を呼び出し，***TGroupBox*** にチェック状態の変化を知らせます。

関連項目：***TGroupBox*** :: ***SelectionChanged***

build static PTStreamable build () ;

TCheckBox(***StreamableInit***)コンストラクタを呼び出します。***TCheckBox*** 型のオブジェクトを構築した後で，そのデータメンバがストリームから読み込まれます。

Check void Check () ;

SetCheck を呼び出してチェックボックスをチェック状態にします。

関連項目：***TCheckBox*** :: ***SetCheck***

GetCheck WORD GetCheck () ;

チェックボックスが非チェック状態であれば ***BF_UNCHECKED*** を返し，チェック状態であれば ***BF_CHECKED*** を，無効状態(淡色表示)であれば ***BF_GRAYED*** を返します。

SetCheck void SetCheck (WORD CheckFlag) ;

チェックボックスを *CheckFlag* で指定された状態にします。*CheckFlag* が ***BF_CHECKED*** であればチェック状態，***BF_UNCHECKED*** であれば非チェック状態，***BF_GRAYED*** であれば無効状態（淡色表示）になります。チェックボックスがグループに属していればグループにも選択内容の変更を知らせます。

関連項目：***TGroupBox*** :: ***SelectionChanged***

read virtual Pvoid read (Ripstream is) ; protected

TWindow :: ***read*** を呼び出して，基本クラス ***TWindow*** のオブジェクトを読み込ませます。その後で ***GetSiblingPtr*** を使って ***Group*** を読み込みます。

Toggle void Toggle () ;

チェックボックスのチェック状態を切り換えます。2状態チェックボックスではチェック状態と非チェック状態を切り換え，3状態チェックボックスではチェック状態，非チェック状態，無効状態（淡色表示）を切り換えます。

関連項目：***TCheckBox*** : : ***SetCheck***

Transfer virtual WORD Transfer (Pvoid DataPtr, WORD TransferFlag) ;

チェックボックスのチェック状態を ***WORD*** 型値（***BF_CHECKED, BF_UNCHECKED, BF_GRAYED*** のいずれか）で，*DataPtr* で指定された転送バッファとの間で転送します。*TranferFlag* が ***TF_GETDATA*** であれば，チェックボックスの状態データはバッファに転送されます。*TransferFlag* が ***TF_SETDATE*** であれば，転送バッファに格納されている状態がチェックボックスにセットされます。***Transfer*** は転送データの大きさをバイト数で返します。転送を行なわずにデータの大きさだけを取得するには，*TransferFlag* として ***TF_SIZEDATA*** を渡します。

Uncheck void Uncheck () ;

SetCheck を呼び出してチェックボックスを非チェック状態にします。

関連項目：***TCheckBox*** : : ***SetCheck***

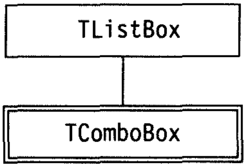
write virtual void write (Ropstream os) ; protected

TWindow : : ***write*** を呼び出し，基本クラス ***TWindow*** のオブジェクトを書き込ませます。その後で ***PutSiblingPtr*** を使って ***Group*** を書き込みます。

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。

TComboBox　　COMBOBOX.H



Windowsのコンボボックスインターフェース要素を表わすインターフェースオブジェクトです。親の ***TWindow*** でコンボボックスコントロールを作成するときは ***TComboBox*** を使わなければなりません。また ***TComboBox*** を使うと，アプリケーションと ***TDialog*** のコンボボックスコントロール間の通信が簡単になります。このオブジェクトの大部分の動作は ***TListBox*** から継承しています。

コンボボックスには，**シンプル**，**ドロップダウン**，**ドロップダウンリスト**の3種類があります。それぞれ ***CBS_SIMPLE***，***CBS_DROPDOWN***，***CBS_DROPDOWNLIST*** の各スタイル定数によって指定されます。***TComboBox*** のコンストラクタにいずれかの定数を渡すと，作成するコンボボックス要素の種類がWindowsに通知されます。

データメンバ

TextLen　WORD TextLen ;

TextLen - 1はコンボボックスの編集領域にある文字バッファの最大長です。***TComboBox*** のコンストラクタによって設定されます。

メンバ関数

constructor　TComboBox (PTWindowsObject AParent, int AnId, int X, int Y, int W, int H,
DWORD AStyle, WORD ATextLen,
PTModule AModule = NULL) ;

渡された親ウィンドウ（*AParent*），コントロールID（*AnId*），親ウィンドウ

のクライアント領域の原点を基準とする座標（*X*, *Y*），幅（*W*），高さ（*H*），スタイル（*AStyle*），テキスト長（*ATextLen*）を使ってコンボボックスオブジェクトを構築します。

同様の引数をとる ***TListBox*** のコンストラクタを呼び出し，***Attr.Style*** を以下のように設定します。

```
Attr.Style = WS_CHILD | WS_VISIBLE | WS_GROUP |
             WS_TABSTOP | CBS_SORT | CBS_AUTOHSCROLL |
             WS_VSCROLL | AStyle ;
```

AStyle には，***CBS_SIMPLE, CBS_DROPDOWN, CBS_DROPDOWNLIST, CBS_OWNERDRAWFIXED, CBS_OWNERDRAWVARIABLE*** のいずれかのコンボボックススタイル定数を設定します。

関連項目：***TComboBox*** : : ***TextLen, TListBox*** : : ***TListBox***

constructor TComboBox (PTWindowsObject AParent, int ResourceId, WORD ATextLen,
PTModule AModule = NULL) ;

TDialog のコンボボックスコントロールに対応する ***TComboBox*** オブジェクトを構築します。同様の引数をとる ***TListBox*** を呼び出し，メンバ ***TextLen*** に *ATextLen* をセットします。

引数 *ResourceID* は，ユーザが定義するコンボボックスリソースに必ず対応させてください。

関連項目：***TListBox*** : : ***TListBox***

constructor TComboBox (StreamableInit) ; protected

TComboBox のストリームコンストラクタです。***TListBox*** のストリームコンストラクタを呼び出します。入力ストリームからのデータを使って ***TComboBox*** のインスタンスを構築するときに呼び出されます。

関連項目：***TListBox*** : : ***TListBox***

build static PTStreamable build () ;

TComboBox(StreamableInit) コンストラクタを呼び出します。***TComboBox*** 型のオブジェクトを構築した後で，そのデータメンバがストリームから読み込まれます。

Clear void Clear() ;

対応するエディットコントロールのテキストをクリアします。

GetClassName virtual LPSTR GetClassName () ; protected

TComboBox の Windows 登録クラス名「COMBOBOX」を返します。

GetEditSel int GetEditSel (Rint StartPos, Rint Endpos) ;

対応するエディットコントロールで選択されているテキストの先頭位置と終端位置を指定された参照引数で返します。エディットコントロールのないコンボボックスの場合は ***CB_ERR*** を返します。

GetMsgID virtual WORD GetMsgID (WORD AnId) ; protected

渡された ObjectWindows メッセージ識別子に対応する Windows コンボボックスメッセージの識別子を返します。***TComboBox*** のインスタンスが ***TListBox*** のメンバ関数の多くを継承できるようにします。

GetText int GetText (LPSTR AString, int MaxChars) ;

指定された文字列に対応するエディットコントロールのテキスト（***MaxChars*** まで）をコピーします。コピーした文字数を返します。

GetTextLen int GetTextLen () ;

対応するエディットコントロールのテキスト長を返します。

HideList void HideList () ;

ドロップダウンコンボボックスまたはドロップダウンリストコンボボックスのリストを非表示にします。

関連項目：***TComboBox*** : : ***ShowList***

nameOf virtual Pchar nameOf () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TComboBox*** のクラス識別文字列である「TComboBox」を返します。

read virtual Pvoid read (Ripstream is) ; protected

TListBox : : ***read*** を呼び出して基本クラス ***TListBox*** のオブジェクトを読

み込ませます。その後で ***TextLen*** を読み込みます。

SetEditSel int SetEditSel (int StartPos, int EndPos) ;

コンボボックスのエディットコントロールにある文字のうち，*StartPos* と *EndPos* の間にある文字を選択します。エディットコントロールがないコンボボックスの場合は ***CB_ERR*** を返します。

SetText void SetText (LPSTR AString) ;

渡された *AString* で始まる最初の文字列をリストボックスから選択します。該当する文字列がなければ，対応するエディットコントロールの内容として，渡された文字列をセットして選択します。

SetupWindow virtual void SetupWindow () ; protected

エディットコントロールの最大テキスト長を ***TextLen*** - 1 とするコンボボックスを設定します。

ShowList void ShowList () ;

ドロップダウンコンボボックスまたはドロップダウンリストコンボボックスのリストを表示します。

関連項目：***TComboBox*** : : ***HideList***

Transfer virtual WORD Transfer (Pvoid DataPtr, WORD TransferFlag) ;

コンボボックスの項目と選択内容を転送バッファとの間で転送します。*TransferFlag* として ***TF_SETDATA*** が渡されれば転送バッファに転送され，***TF_GETDATA*** が渡されれば転送バッファからコンボボックスに転送されます。***TransferBuffer*** は，コンボボックスのデータ転送バッファとして使われる ***PTComboBoxData*** オブジェクトを指すものとします。

戻り値は ***PTComboBoxData***（構造体を指すポインタ，構造体そのものではない）のサイズです。データを転送せずにサイズだけを取得するには，*TransferFlag* として ***TF_SIZEDATA*** を渡します。

➡ 転送バッファには，構造体を指すポインタを使わなければなりません。構造体のコピーを転送バッファに入れることはできません。構造体を転送バッファとしても使えません。

関連項目：***TListBox*** : : ***Transfer***

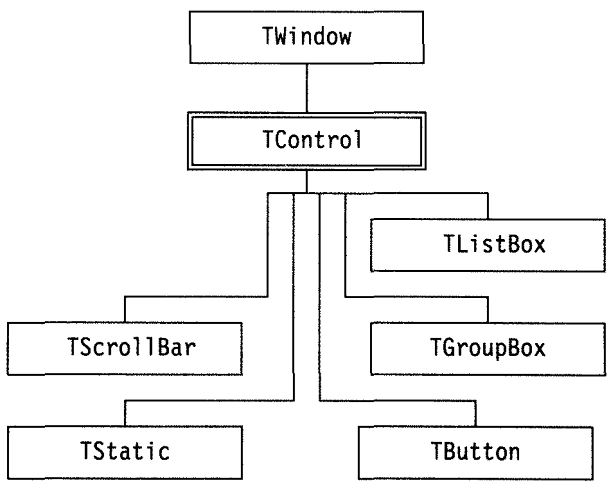
write Virtual void write (Ropstream os) ; protected

TListBox :: *write* を呼び出して基本クラス *TListBox* のオブジェクトを書き込ませます。その後で *TextLen* を書き込みます。

関連関数

operator＜＜と operator＞＞については，13ページを参照してください。

TControl CONTROL.H



TScrollBar や *TButton* などの派生コントロールクラスを統合します。派生クラスのコントロールオブジェクトは，Windows のコントロールインターフェース要素を表わすのに使います。親の *TWindow* でコントロールを作成するにはコントロールオブジェクトを使わなければなりません。またコントロールオブジェクトを使うと，アプリケーションと *TDialog* のコントロール間の通信が簡単になります。

メンバ関数

constructor TControl (PTWindowsObject AParent, int AnId, LPSTR ATitle, int X, int Y, int W, int H, PTModule AModule = NULL) ;

TWindow のコンストラクタを呼び出し，親ウィンドウ（*AParent*），タイトルテキスト（*ATitle*），*AModule* を渡します。渡された引数のウィンドウ識別子（*AnID*），親ウィンドウのクライアント領域の原点を基準とする座標（*X*, *Y*），幅（*W*），高さ（*H*）を使って ***Attr*** を設定します。***Attr.Style*** に ***WS_CHILD*** | ***WS_VISIBLE*** | ***WS_GROUP*** | ***WS_TABSTOP*** をセットします。

関連項目：***TWindow*** :: ***TWindow***

constructor TControl (PTWindowsObject AParent, int ResourceId, PTModule AModule = NULL) ;

TDialog のインターフェースコントロールに対応するオブジェクトを構築します。***TWindow*** のコンストラクタを呼び出し，さらに ***EnableTranfer*** を呼び出してデータ転送機構を有効にします。

引数 *ResourceID* はユーザが定義するコントロールインターフェースリソースに必ず対応させてください。

関連項目：***TWindow*** :: ***TWindow, TWindowsObject*** :: ***EnableTransfer***

constructor TControl (StreamableInit) ; protected

TControl のストリームコンストラクタです。***TWindow*** のコンストラクタを呼び出します。入力ストリームからのデータを使って ***TControl*** のインスタンスを構築するときに呼び出されます。

関連項目：***TWindow*** :: ***TWindow***

GetId virtual int GetId () ;

ウィンドウ識別子 ***Attr.Id*** を返します。

ODADrawEntire virtual void ODADrawEntire (DRAWITEMSTRUCT _FAR & DrawInfo) ;

protected

描画可能コントロールを描画するときにこのコントロールに通知が送られると，デフォルトで ***Parent - >DrawItem*** を呼び出します。描画可能コントロールで定義し直すと，描画方法を指定できます。

関連項目：***TWindowsObject*** :: ***DrawItem, TControl*** :: ***WMDrawItem***

ODAFocus virtual void ODAFocus (DRAWITEMSTRUCT _FAR & DrawInfo) ;

protected

描画可能コントロールのフォーカスが移動したときにこのコントロールに通知が送られると，デフォルトで ***Parent - >DrawItem*** を呼び出します。描画可能コントロールで定義し直すと，フォーカスを失ったときまたは受け取ったときの描画方法を指定できます。

関連項目：***TWindowsObject*** :: ***DrawItem***

ODASelect virtual void ODASelect (DRAWITEMSTRUCT _FAR & DrawInfo) ;

protected

描画可能コントロールの選択状態が変更したときにこのコントロールに通知が送られると，デフォルトで ***Parent - >DrawItem*** を呼び出します。描画可能コントロールで定義し直すと，選択状態が変わったときの描画方法を指定できます。

関連項目：***TWindowsObject*** :: ***DrawItem, TWindow*** :: ***WMDrawItem***

WMDrawItem virtual void WMDrawItem (RTMessage Msg) = [WM_FIRST + WM_DRAWITEM] ;

描画可能コントロールの描画が必要になったときにこのコントロールに送られる ***WM_DRAWITEM*** メッセージに応答します。コントロール全体の描画が必要であれば，***TControl*** :: ***WMDrawItem*** は ***ODADrawEntire*** を呼び出します。コントロールの選択状態が変ったときは ***ODASelect*** を呼び出します。フォーカスが移動したときは ***ODAFocus*** を呼び出します。

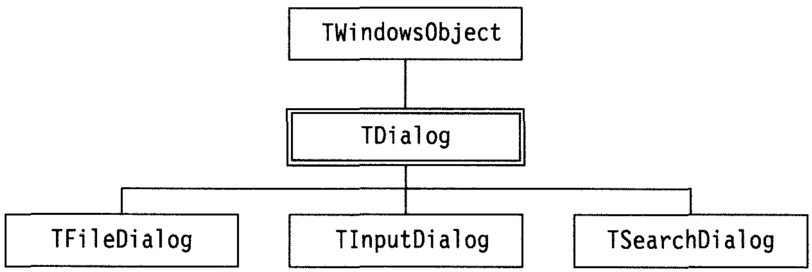
関連項目：***TControl*** :: ***ODADrawEntire, TControl*** :: ***ODASelect, TControl*** :: ***ODAFocus, TWindowsObject*** :: ***DrawItem***

WMPaint　virtual void WMPaint (RTMessage Msg) ＝ ［WM_FIRST ＋ WM_PAINT］；

コントロールに定義済みの Windows クラスがあれば，Windows 指定の描画を行なわせるため ***DefWndProc*** を呼び出します。定義済みの Windows クラスがなければ，***TWindow*** :: ***WMPaint*** を呼び出します。

関連項目：***TWindow*** :: ***WMPaint***

TDialog　　DIALOG.H



モードつきダイアログボックスとモードなしダイアログボックスのインターフェース要素を表わします。***TDialog*** には対応するリソース定義があり，コントロールの配置と形状が記述されています。このリソース定義の識別子は ***TDialog*** オブジェクトのコンストラクタに渡されます。

モードつきインターフェース要素に対応づけるにはメンバ関数 ***Execute*** を呼び出し，モードなしインターフェース要素に対応づけるにはメンバ関数 ***Create*** を呼び出します。ただし，通常はこの 2 つのメンバ関数を ObjectWindows のアプリケーションから直接呼び出しません。***TModule*** のメンバ関数 ***ExecDialog*** と ***MakeWindow*** をかわりに呼び出し，***TDialog*** の妥当性をチェックしてから対応づけを行ないます。

➡ モードつきダイアログをオープンしている間は，親ウィンドウでの操作が無効になります。

データメンバ

Attr TDialogAttr Attr ;

ダイアログボックスの作成属性を保持します。***TDialogAttr*** の定義は，第3章『**ObjectWindows のその他の構成要素**』にあります。

IsModal BOOL IsModal ;

ダイアログボックスがモードつきの場合は真，モードなしの場合は偽になります。

メンバ関数

constructor TDialog (PTWindowsObject AParent, LPSTR AName,
PTModule AModule = NULL) ;

TWindowsObject のコンストラクタを呼び出し，*AParent* と *AModule* を渡します。***DisableAutoCreate*** を呼び出し，***TDialog*** が親ウィンドウと同時に自動的に作成されないようにします。***Title*** に初期値 -1 を設定します。***AName*** のコピーを ***Attr.Name*** に代入します。この *AName* はリソース定義ファイルで定義しなければならない文字列リソース識別子です。***Attr.Param*** を 0 に初期化し，***IsModal*** に偽をセットします。

関連項目：***TWindowsObject*** : : ***TWindowsObject,***
TWindowsObject : : ***DisableAutoCreate***

constructor TDialog (PTWindowsObject AParent, int ResourceId,
PTModule AModule = NULL) ;

TWindowsObject のコンストラクタを呼び出し，*AParent* と *AModule* を渡します。***DisableAutoCreate*** を呼び出し，***TDialog*** が親と同時に自動的に作成されないようにします。***Title*** の初期値 -1 を設定します。ダイアログの整数リソース識別子を ***Attr.Name*** に設定します。この識別子はリソースファイルのダイアログリソース定義に必ず対応させなければなりません。***Attr.Param*** を 0 に初期化し，***IsModal*** に偽をセットします。

関連項目：*TWindowsObject::TWindowsObject, TWindowsObject::DisableAutoCreate*

constructor TDialog (StreamableInit) ; protected

TDialog のストリームコンストラクタです。*TWindowsObject* のストリームコンストラクタを呼び出します。入力ストリームからのデータを使って *TDialog* のインスタンスを構築するときに呼び出されます。

関連項目：*TWindowsObject::TWindowsObject*

destructor virtual ~TDialog () ;

Attr.Name が文字列であり，整数リソース識別子でなければ，*TDialog* 名で確保されたファイル名 *Attr.Name* のメモリを解放します。

関連項目：*TWindowsObject::~TWindowsObject*

build static PTStreamable build () ;

*TDialog(StreamableInit)*コンストラクタを呼び出します。*TDialog* 型のオブジェクトを構築した後でそのデータメンバがストリームから読み込まれます。

Cancel virtual void Cancel (RTMessage Msg) = [ID_FIRST + IDCANCEL] ; protected

ダイアログボックスの Cancel ボタンがクリックされると自動的に応答し，*CloseWindow*(int)を呼び出して *IDCANCEL* を渡します。

関連項目：*TDialog::CloseWindow*

CloseWindow virtual void CloseWindow (int ARetValue) ;

ダイアログボックスを条件的にクローズします。this がモードなしダイアログボックスであれば，*TWindowsObject::CloseWindow* を呼び出します。this がモードつきダイアログボックスであれば，*CanClose* を呼び出します。*CanClose* が真を返せば，*TransferData* を呼び出してダイアログボックスデータを転送させてから *ShutDownWindow* を呼び出して *ARetValue* を渡します。

関連項目：*TWindowsObject::CloseWindow, TDialog::CanClose, TDialog::TransferData, TDialog::ShutDownWindow*

CloseWindow virtual void CloseWindow () ;

ダイアログボックスを条件的にクローズします。this がモードなしダイアログボックスであれば，***TWindowsObject*** : : ***CloseWindow*** を呼び出します。this がモードつきダイアログボックスであれば，***CloseWindow*** を呼び出し，整数 *ARetValue* として ***IDCANCEL*** を渡します。

関連項目：***TWindowsObject*** : : ***CloseWindow***

Create virtual BOOL Create () ;

TDialog オブジェクトに対応するモードなしダイアログボックスのインターフェース要素を作成します。ただし，***Status*** が 0 以外の場合，この対応づけは行なわれません。***DisableAutoCreate*** を呼び出し，すべての子ウィンドウが自動的に作成されないようにします。***EnableKBHandler*** を呼び出してキーボード処理を有効にさせます。ダイアログのすべての子ウィンドウを登録して，カスタムコントロールをサポートできるようにします。Windows 関数 ***CreateDialogParam*** を呼び出してダイアログボックスを作成させます。

成功すれば ***Create*** は真を返し，失敗すれば ***Status*** に ***EM_INVALIDWINDOW*** が設定され，***Create*** は偽を返します。

☞ モードなしダイアログボックスを作成するには ***TModule*** : : ***MakeWindow*** を使う方が無難です。

関連項目：***TDialog*** : : ***Execute, TModule*** : : ***MakeWindow,***
TWindowsObject : : ***DisableAutoCreate,***
TWindowsObject : : ***EnableKBHandler***

Destroy virtual void Destroy (int ARetValue) ;

TDialog に対応するインターフェース要素を破棄します。this がモードなしダイアログボックスの場合は，***TWindowsObject*** : : ***Destroy*** を呼び出します。this がモードつきダイアログボックスの場合は，すべての子ウィンドウに対して ***EnableAutoCreate*** を呼び出します。さらに Windows 関数 ***EndDialog*** を呼び出し，ダイアログの実行結果の戻り値として *ARetValue* を渡します。

関連項目：***TWindowsObject*** : : ***Destroy,***
TWindowsObject : : ***EnableAutoCreate***

Destroy virtual void Destroy () ;

TDialog に対応するインターフェース要素を破棄します。this がモードなしダイアログボックスの場合は，***TWindowsObject*** : : ***Destroy*** を呼び出します。this がモードつきダイアログボックスの場合は，***Destroy*** を呼び出し，整数型 *ARetValue* として ***IDCANCEL*** を渡します。

関連項目：***TWindowsObject*** : : ***Destroy***

Execute virtual int Execute () ;

TDialog オブジェクトに対応するモードつきダイアログボックスのインターフェース要素を作成して実行します。メンバ ***Status*** が 0 以外の場合，この対応づけは行なわれません。対応づけに成功すると，***TDialog*** がクローズするまで実行を続けます。

DisableAutoCreate を呼び出し，すべての子ウィンドウが自動的に作成されないようにします。***EnableKBHandler*** を呼び出してキーボード処理を有効にします。すべてのダイアログの子ウィンドウを登録してカスタムコントロールをサポートできるようにします。Windows 関数 ***DialogBoxParam*** を呼び出してダイアログボックスを実行させます。

☞ モードつきダイアログボックスを作成するには ***TModule*** : : ***ExecWindow*** を使う方が無難です。

関連項目：***TModule*** : : ***ExecDialog,***
TWindowsObject : : ***DisableAutoCreate,***
TWindowsObject : : ***EnableKBHandler***

GetClassName virtual LPSTR GetClassName () ; **protected**

ダイアログボックスのデフォルトの Windows クラス名を返します。モードつきダイアログボックスの場合はこのクラス名を使わなければなりません。モードなしダイアログボックスの場合は，デフォルトの ObjectWindows クラス名である「OWLDialog」を返します。

GetItemHandle HWND GetItemHandle (int DlgItemID) ;

渡された ID *DlgItemID* によって特定されたダイアログボックスのウィンドウハンドルを返します。

GetWindowClass virtual void GetWindowClass (WNDCLASS _FAR & AWndClass) ; protected

TDialog のデフォルト登録属性を ***AWndClass*** に入れます。***TDialog*** のデフォルトのアイコンとカーソルは，アプリケーションの標準アイコンと矢印カーソルになります。デフォルトの登録属性を変更するには，このメンバ関数と ***GetClassName*** を ***TDialog*** の派生クラスで定義し直します。メンバ関数 ***GetWindowClass*** で ***TDialog*** が設定するデフォルトを変更するときは，あらかじめ必ず ***TDialog*** : : ***GetWindowClass*** を呼び出してください。

関連項目：***TWindowsObject*** : : ***GetClassName***

isA virtual classType isA () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TDialog*** のクラス識別子である ***dialogClass*** を返します。

nameOf virtual Pchar nameOf () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TDialog*** のクラス識別文字列である「TDialog」を返します。

Ok virtual void Ok (RTMessage Msg) = [ID_FIRST + IDOK] ; protected

ダイアログボックスの OK ボタン（識別子 ***IDOK***）がクリックされると，***CloseWindow*** を呼び出して ***IDOK*** を渡します。

関連項目：***TDialog*** : : ***CloseWindow***

read virtual Pvoid read (Ripstream is) ; protected

TWindowsObject : : ***read*** を呼び出して，基本クラス ***TWindowsObject*** のオブジェクトを読み込ませます。その後で ***Attr.Name*** と ***IsModal*** を読み込みます。

SendDlgItemMsg DWORD SendDlgItemMsg (int DlgItemID, WORD AMsg, WORD WParam, DWORD LParam) ;

AMsg で特定された Windows コントロールメッセージを，***DlgItemID*** で特定されたダイアログボックスのコントロールに送ります。***WParam*** と ***LParam*** はコントロールメッセージの引数になります。***SendDlgItemMsg*** はコントロールからの戻り値を返します。コントロール ID が無効の場合は 0 を返します。

SetCaption void SetCaption (LPSTR ATitle) ;

ATitle が -1 でなければ，***TWindowsObject*** :: ***SetCaption*** を呼び出します。

関連項目：***TDialog*** :: ***SetupWindow, TWindowsObject*** :: ***SetCaption***

SetupWindow virtual void SetupWindow () ; protected

SetCaption（***Title*** の設定）と ***TWindowsObject*** :: ***SetupWindow*** を呼び出して，ダイアログボックスを設定します。

関連項目：***TDialog*** :: ***SetCaption, TWindowsObject*** :: ***SetupWindow***

ShutDownWindow virtual void ShutDownWindow () ;

ダイアログボックスをシャットダウンします。this がモードなしダイアログボックスであれば，***TWindowsObject*** :: ***ShutDownWindow*** を呼び出します。this がモードつきダイアログボックスであれば，***Destroy*** を呼び出して ***IDCANCEL*** を渡します。

関連項目：***TWindowsObject*** :: ***ShutDownWindow, TDialog*** :: ***Destroy***

ShutDownWindow virtual void ShutDownWindow (int ARetValue) ;

ダイアログボックスをシャットダウンします。this がモードなしダイアログボックスであれば，***TWindowsObject*** :: ***ShutDownWindow*** を呼び出します。this がモードつきダイアログボックスであれば，***Destroy*** を呼び出して *ARetValue* を渡します。

関連項目：***TWindowsObject*** :: ***ShutDownWindow, TDialog*** :: ***Destroy***

WMClose virtual void WMClose (RTMessage Msg) = [WM_FIRST + WM_CLOSE] ; protected

WM_CLOSE メッセージが送られてくると，ウィンドウをシャットダウンします。

関連項目：***TDialog*** :: ***ShutDownWindow***

WMInitDialog virtual void WMInitDialog (RTMessage Msg)
= [WM_FIRST + WM_INITDIALOG] ; protected

ダイアログボックスの表示直前に自動的に呼び出されます。***SetupWindow*** を呼び出して，ダイアログボックスまたはそのコントロールに必要な初期化を行なわせます。

関連項目：***TWindowsObject*** :: ***SetupWindow***

WMQueryEndSession virtual void WMQueryEndSession(RTMessage Msg)
= [WM_FIRST + WM_QUERYENDSESSION] ; protected

Windows がクローズしようとすると応答します。this がメインウィンドウであれば，***GetApplication***() －＞***CanClose*** を呼び出します。メインウィンドウでなければ，this －＞***CanClose*** を呼び出します。***CanClose*** を呼び出して真が返れば（シャットダウンしてもよければ），0 を返して ***Msg.Result*** を設定します。***CanClose*** を呼び出して偽が返れば，ダイアログボックスは 1 を返します。

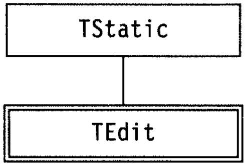
Windows の ***WM_QUERYENDSESSION*** とは逆の動作のことです。

write virtual void write (Ropstream os) ; protected

TWindowsObject :: ***write*** を呼び出して基本クラス ***TWindowsObject*** のオブジェクトを書き込ませます。その後で ***Attr.Name*** と ***IsModal*** を書き込みます。

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。



Windows のエディットコントロールのインターフェース要素を表わすインターフェースオブジェクトです。親の ***TWindow*** でエディットコントロールを作成するときは ***TEdit*** のオブジェクトを必ず使わなければなりません。また ***TEdit*** を使うとアプリケーションと ***TDialog*** のエディットコントロール間の通信が簡単になります。

エディットコントロールオブジェクトには単一行と複数行の 2 つのスタイルがあります。複数行エディットコントロールでは複数行のテキストの編集が可能です。なお，エディットコントロールの最初の文字位置は 0 になります。複数行のエディットコントロールでは，位置番号は行単位の追番になります。改行は 2 文字分として数えます。

TEdit のメンバ関数の大部分はエディットコントロールのテキストを管理するものですが，他に自動応答用のメンバ関数もあります。自動応答メンバ関数は，ユーザが親ウィンドウのメニュー項目である切り抜き，複写，貼り付けを選択すると，自動的に応答します。***TEdit*** の基本クラス ***TStatic*** からは 2 つの重要なメンバ関数 ***GetText*** と ***SetText*** を継承しています。

メンバ関数

constructor TEdit (PTWindowObject AParent, int AnId, LPSTR AText, int X, intY,
int W; int H, WORD ATextLen, BOOL Multiline,
PTModule AModule = NULL);

親ウィンドウの *AParent* を使ってエディットコシトロールのオブジェクトを構築します。エディットコントロールの作成属性を設定し，そのデータメンバ ***Attr*** に，渡されたコントロール ID（*AnId*），親ウィンドウのクライアント領域の原点を基準とする座標（*X, Y*），幅（*W*），高さ（*H*）を入れます。

テキストバッファ長（*ATextLen*）が0か1であれば，入力できる文字数に対し明示的な制限はありません。それ以外の場合は，*ATextLen* - 1までの文字数を入力できます。デフォルトでは，エディットコントロールに枠がつき，その最初のテキスト（*ATitle*）は左詰めになります。複数行エディットコントロールには水平と垂直のスクロールバーがつきます。

constructor TEdit (PTWindowsObjectAParent, int ResourceId, WORD ATextLen,
PTModule AModule = NULL) :

TDialog のエディットコントロールに対応する ***TEdit*** オブジェクトを構築します。同じ引数を使う ***TStatic*** のコンストラクタを呼び出します。引数 ***ResourceId*** は，ユーザが定義するエディットコントロールに必ず対応させてください。***EnableTransfer*** を呼び出してデータ転送機構を有効にさせます。

関連項目：***TStatic*** : : ***TStatic***

constructor TEdit (StreamableInit) ; protected

TEdit のストリームコンストラクタです。***TStatic*** のストリームコンストラクタを呼び出します。入力ストリームからのデータを使って ***TEdit*** のインスタンスを構築するときに呼び出されます。

関連項目：***TStatic*** : : ***TStatic***

build static PTStreamable build () ;

TEdit(StreamableInit)コンストラクタを呼び出します。***TEdit*** 型のオブジェクトを構築した後でそのデータメンバがストリームから読み込まれます。

CanUndo BOOL CanUndo () ;

最終の編集をやり直せる場合は真を返します。

関連項目：***TEdit*** : : ***Undo***

ClearModify void ClearModify () ;

エディットコントロールの変更フラグを再設定して ***IsModified*** に偽を返させます。このフラグはテキストが変更されると真になります。

関連項目：***TEdit*** : : ***IsModified***

CMEditClear virtual void CMEditClear (RTMessage Msg)
= [CM_FIRST + CM_EDITCLEAR] ; protected

メニュー ID が ***CM_EDITCLEAR*** のメニュー選択に自動的に応答し，***Clear*** を呼び出します。

関連項目：***TEdit*** : : ***Clear***

TE Class

CMEditCopy virtual void CMEditCopy (RTMessage Msg)
= [CM_FIRST + CM_EDITCOPY] ; protected

メニュー ID が ***CM_EDITCOPY*** のメニュー選択に自動的に応答し，***Copy*** を呼び出します。

関連項目：***TEdit*** : : ***Copy***

CMEditCut virtual void CMEditCut (RTMessage Msg)
= [CM_FIRST + CM_EDITCUT] ; protected

メニュー ID が ***CM_EDITCUT*** のメニュー選択に自動的に応答し，***Cut*** を呼び出します。

関連項目：***TEdit*** : : ***Cut***

CMEditDelete virtual void CMEditDelete (RTMessage Msg)
= [CM_FIRST + CM_EDITDELETE] ; protected

メニュー ID が ***CM_EDITDELETE*** のメニュー選択に自動的に応答し，***DeleteSelection*** を呼び出します。

関連項目：***TEdit*** : : ***DeleteSelection***

CMEditPaste virtual void CMEditPaste (RTMessage Msg)
= [CM_FIRST + CM_EDITPASTE] ; protected

メニュー ID が ***CM_EDITPASTE*** のメニュー選択に自動的に応答し，***Paste*** を呼び出します。

関連項目：***TEdit*** : : ***Paste***

CMEditUndo virtual void CMEditUndo (RTMessage Msg)
= [CM_FIRST + CM_EDITUNDO] ; protected

メニュー ID が ***CM_EDITUNDO*** のメニュー選択に自動応答し，***Undo*** を呼び出します。

関連項目：***TEdit*** : : ***Undo***

Copy void Copy () ;

現在選択されているテキストをクリップボードに複写します。

関連項目：***TEdit*** : : ***CMEditCopy***

Cut void Cut () ;

現在選択されているテキストを削除して，クリップボードに複写します。

関連項目：***TEdit*** : : ***CMEditCut***

DeleteLine BOOL DeleteLine (int LineNumber) ;

複数行エディットコントロールから *LineNumber* で指定された行のテキストを削除します。－1 が渡されれば，現在の行を削除します。***DeleteLine*** は改行を削除しないので他の行はそのままです。行の削除に成功すると真を返します。*LineNumber* が－1 以外か範囲外の場合，またはエラーが発生した場合は偽を返します。

DeleteSelection BOOL DeleteSelection () ;

現在選択されているテキストを削除します。選択されているテキストがなければ偽を返します。

関連項目：***TEdit*** : : ***CMEditDelete***

DeleteSubText BOOL DeleteSubText (int StartPos, int EndPos) ;

StartPos で指定された開始位置と *EndPos* で指定された終了位置の間のテキストを削除します。成功すれば真を返します。

ENErrSpace virtual void ENErrSpace (RTMessage Msg)
= [NF_FIRST + EN_ERRSPACE] ; protected

エディットコントロールが追加メモリの確保に失敗したときに送られるエラー通知メッセージに応答し，ビープ音を鳴らします。

GetClassName virtual LPSTR GetClassName () ; protected

TEdit のウィンドウ登録クラス名である「EDIT」を返します。

GetLine BOOL GetLine (LPSTR ATextString, int StrSize, int LineNumber) ;

行番号で指定された行のテキストをエディットコントロールから取得して，ヌルで終わる文字列 *ATextString* で返します。*StrSize* は取得すべき文字数を示します。テキストを取得できなかった場合，または渡されたバッファが小さすぎた場合は偽を返します。

関連項目：***TStatic* : : *GetText*, *TEdit* : : *GetNumLines*, *TEdit* : : *LineLength***

GetLineFromPos int GetLineFromPos (int CharPos) ;

複数行エディットコントロールから，*CharPos* で指定された文字位置の行番号を返します。*CharPos* が最終の文字位置を超えた場合は最終の行番号を返します。-1が渡されると，最初に選択された文字を含む行の行番号を返します。

GetLineIndex int GetLineIndex (int LineNumber) ;

複数行エディットコントロールから，*LineNumber* で指定された行番号の前の行までの文字数を返します。*LineNumber* が -1 の場合は，キャレットを含む行の行番号を返します。

GetLineLength int GetLineLength (int LineNumber) ;

複数行エディットコントロールから，*LineNumber* で指定された行の文字数を返します。*LineNumber* が -1 の場合は，次のいずれかの値を返します。選択されているテキストがなければ，キャレット位置の行の長さを返します。選択されているテキストが行内にあれば，選択文字数を引いた行の長さを返します。選択されているテキストが複数行にわたる場合は，選択文字数を引いた複数行の長さを返します。

GetNumLines int GetNumLines () ;

複数行エディットコントロールに入力された行数を返します。テキストがないか，エラーが起こった場合は 0 を返します。

GetSelection void GetSelection (Rint StartPos, Rint EndPos) ;

現在選択されているテキストの開始位置と終了位置を取得して，引数 *StartPos* と *EndPos* で返します。***GetSelection*** と ***GetSubText*** を併用すると現在選択されているテキストを取得できます。

関連項目：***TEdit*** : : ***GetSubText***

GetSubText void GetSubText (LPSTR ATextString, int StartPos, int EndPos) ;

StartPos インデックスから *EndPos* インデックスまでのテキストをエディットコントロールから取得して，*ATextString* で返します。

関連項目：***TEdit*** : : ***GetSelection***

Insert void Insert (LPSTR ATextString) ;

ATextString で渡されたテキストを，エディットコントロールの現在のキャレット位置に挿入します。現在選択されているテキストがあれば置き換えます。***Insert*** は ***Paste*** に似ていますが，クリップボードは変更されません。

関連項目：***TEdit*** : : ***Paste***

IsModified BOOL IsModified () ;

ユーザがエディットコントロールのテキストを変更した場合は真を返します。

関連項目：***TEdit*** : : ***ClearModify***

Paste void Paste () ;

クリップボードのテキストをエディットコントロールの現在のキャレット位置に挿入します。

関連項目：***TEdit*** : : ***CMEditPaste***

Scroll void Scroll (int HorizontalUnit, int VerticalUnit) ;

複数行エディットコントロールを *HorizontalUnit* と *VerticalUnit* で指定された文字数分だけ水平方向と垂直方向にスクロールします。正の値で右方向と下方へスクロールし，負の値で左方向と上方へスクロールします。

Search int Search (int StartPos, LPSTR AText, BOOL CaseSensitive) ;

渡されたテキストの大文字小文字を区別してまたは区別せずに検索します。テキストが見つかれば選択して，テキストの位置を返します。見つからなければ，-1を返します。*StartPos* で-1が渡されると，現在選択されているテキストの最後から検索を始めます。選択されているテキストがなければ，キャレットの位置から始めます。

SetSelection BOOL SetSelection (int StartPos, int EndPos) ;

StartPos と *EndPos* で指定された位置の間にあるテキストを選択します。ただし，*EndPos* にある文字は含めません。

SetupWindow virtual void SetupWindow () ; protected

データメンバ ***TextLen*** が0以外の場合は，エディットコントロールに入力できる文字数を ***TextLen*** -1までに制限します。

関連項目：***TStatic*** :: ***TextLen***

Undo void Undo () ;

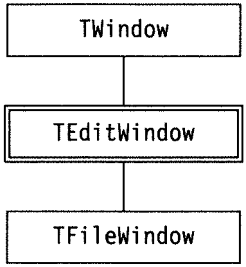
最終の編集をアンドゥします。

関連項目：***TEdit*** :: ***CanUndo***, ***TEdit*** :: ***CMEditUndo***

関連関数

operator＜＜と operator＞＞については，13ページを参照してください。

TEditWindow　EDITWND.H



テキストの入力と編集専用のウィンドウクラスを定義します。このクラスではテキスト編集を管理するフルウィンドウのエディットコントロールが規定されます。下位型の ***TFileWindow*** はファイルの編集用です。

データメンバ

Editor　PTEdit Editor ;

エディットウィンドウにテキスト編集機能を持たせるフルウィンドウの複数行エディットコントロールを指すポインタです。

IsReplaceOp　BOOL IsReplaceOp ;

置換操作を次の検索操作と同時に行なう場合は真を保持します。

SearchStruct　TSearchStruct SearchStruct ;

TSearchDialog で使う転送バッファです。

関連項目：***TSearchDialog***

メンバ関数

constructor TEditWindow(PTWindowsObject AParent, LPSTR ATitle,
PTModule AModule = NULL);

Editor エディットコントロールのインスタンスを構築し，***SearchStruct*** をゼロにし，***IsReplaceOp*** をヌルに設定します。*ATitle* はタイトルバーのウィンドウタイトルを指定し，***TWindow*** コンストラクタに渡されます。

constructor TEditWindow (StreamableInit); protected

TEditWindow のストリームコンストラクタです。***TWindow*** のストリームコンストラクタを呼び出します。入力ストリームのデータで ***TEditWindow*** のインスタンスを構築するときに呼び出されます。

build static PTStreamable build ();

TEditWindow(StreamableInit) のコンストラクタを呼び出します。型 ***TEditWindow*** のオブジェクトを構築した後にデータメンバをストリームから読み込みます。

CMEditFind virtual void CMEditFind (RTMessage Msg)
= [CM_FIRST + CM_EDITFIND]; protected

Find メニュー項目が選択されると検索ダイアログボックスを実行し，ユーザから ***SearchStruct*** に転送された検索対象のテキストとオプションを取得します。***IsReplaceOp*** を偽に設定してから ***DoSearch*** を呼び出します。

関連項目：***TEditWindow*** :: ***DoSearch, TSearchStruct***

CMEditFindNext virtual void CMEditFindNext (RTMessage Msg)
= [CM_FIRST + CM_EDITFINDNEXT]; protected

Find Next メニュー項目が選択されると ***DoSearch*** を呼び出します。***SearchStruct*** にはテキストと検索/置換オプションがすでにあるものとみなします。

関連項目：***TEditWindow*** :: ***DoSearch, TSearchStruct***

CMEditReplace virtual void CMEditReplace (RTMessage Msg)
= [CM_FIRST + CM_EDITREPLACE] ; protected

Replace メニュー項目が選択されると ***DoSearch*** を呼び出します。検索ダイアログボックスを実行して，ユーザから ***SearchStruct*** に転送された検索/置換対象のテキストとオプションを取得します(第 3 章『**ObjectWindows のその他の構成要素**』の ***TSearchStruct*** を参照)。***IsReplaceOp*** を真に設定し，***DoSearch*** を呼び出します。

関連項目：***TEditWindow*** : : ***DoSearch***

DoSearch void DoSearch () ;

SearchStruct のテキストとオプションに一致するテキストを ***Editor*** 内で探します。メッセージボックスを表示して，エラーを報告し，テキストの置換をするかどうかの確認応答を取得します（第 3 章『**ObjectWindows のその他の構成要素**』の ***TSearchStruct*** を参照)。

関連項目：***TEdit*** : : ***Search***

read virtual Pvoid read (Pipstream is) ; protected

TWindow : : ***read*** を呼び出して，基本クラス ***TWindow*** のオブジェクトを読み込ませます。さらに，***GetChildPtr*** を呼び出して子ウィンドウのエディットコントロールを読み込ませます。

WMSetFocus virtual void WMSetFocus (RTMessage Msg)
= [WM_FIRST + WM_SETFOCUS] ; protected

WM_SETFOCUS メッセージを受け取ると ***Editor*** エディットコントロールにフォーカスを設定します。

WMSize virtual void WMSize (RTMessage Msg)
= [WM_FIRST + WM_SIZE] ; protected

WM_SIZE メッセージを受け取ると ***Editor*** エディットコントロールを ***TEditWindow*** のクライアント領域に収まるサイズに変更します。

write virtual void write (Ropstream os) ; protected

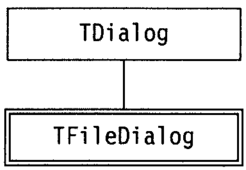
TWindow : : ***write*** を呼び出して，基本クラス ***TWindow*** のオブジェクトを書き出させます。次に，***PutChildPtr*** を呼び出してエディットコントロール子

ウィンドウを書き出させます。

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。

TFileDialog　　FILEDIAL.H



ファイルをオープンまたはセーブするときのファイル名を，ユーザに選択させるためのダイアログボックスを作成します。ダイアログボックスリソース FILEDIAL.DLG に，ファイルのオープン用とセーブ用のダイアログボックスとして，2 つのダイアログボックスが定義されています。

データメンバ

Extension　char Extension［MAXEXT］;

ファイル名の拡張子が入ります。

FilePath　LPSTR FilePath;

ユーザが指定したファイル名を返すバッファを指すポインタです。ファイルダイアログボックスのコンストラクタに渡されるときは，初期ファイルマスクが入っています。

FileSpec　char FileSpec［FILESPEC］;

現在のファイル名が入ります。

PathName char PathName [MAXPATH] ;

現在のファイルパスが入ります。

メンバ関数

constructor TFileDialog (PTWindowsObject AParent, int ResourceId, LPSTR AFilePath, PTModule AModule = NULL) ;

TDialog のコンストラクタを呼び出し，*AParent, ResourceId, AModule* を渡します。***FilePath*** に *AFilePath* を設定します。*AFilePath* が指すバッファの内容をテンプレートとして使い，ファイルリストボックスに書き込みます。***SD_FILEOPEN***(File Open ダイアログの場合) または ***SD_FILESAVE***(File Save ダイアログの場合) を，ダイアログボックスの ***ResourceId*** (整数のリソース識別子) として指定してください。

関連項目：***TDialog*** :: ***TDialog***

constructor TFileDialog (StreamableInit) ; protected

TFileDialog のストリームコンストラクタです。入力ストリームのデータで ***TFileDialog*** のインスタンスを構築するときに呼び出されます。

build static PTStreamable build () ;

TFileDialog (***StreamableInit***)コンストラクタを呼び出します。型 ***TFileDialog*** のオブジェクトを構築してから，データメンバがストリームから読み込まれます。

CanClose virtual BOOL CanClose () ;

ユーザが有効なファイル名を入力すると真を返します。エディットコントロールのテキストを取得して ***PathName*** を更新します。***UpdateListBoxes*** を呼び出し，真が返ると偽を返します。エディットコントロールに無効なファイル名が入力されると偽を返します。

関連項目：***TFileDialog*** :: ***UpdateListBoxes***

HandleDList virtual void HandleDList (RTMessage Msg)
= [ID_FIRST + ID_DLIST] ; protected

ディレクトリリストボックスからのメッセージに応答します。リストボックスの選択内容が変わると，***PathName*** を更新します。選択項目がダブルクリックされると ***UpdateListBoxes*** を呼び出し，それ以外は ***UpdateFileName*** を呼び出します。選択項目がフォーカスを失うと，リストボックスの選択を消去します。

関連項目：***TFileDialog*** : : ***UpdateListBoxes***

HandleFList virtual void HandleFList (RTMessage Msg)
= [ID_FIRST + ID_FLIST] ; protected

ファイルリストボックスのメッセージに応答します。リストボックスの選択が変わると，選択したファイル名で *PathName* を更新し，***UpdateFileName*** を呼び出します。選択項目がダブルクリックされると，ダイアログボックスをクローズします。選択項目がフォーカスを失うと，リストボックスの選択を消去します。

関連項目：***TFileDialog*** : : ***UpdateFileName***

HandleFName virtual void HandleFName (RTMessage Msg)
= [ID_FIRST + ID_FNAME] ; protected

エディットコントロールからのメッセージに応答します。エディットコントロールにテキストがあれば，OK ボタンを使えるようにします。

SelectFileName void SelectFileName () ;

エディットコントロール内のテキストを選択します。また，エディットコントロールにフォーカスを割り当てます。

SetupWindow virtual void SetupWindow () ; protected

ファイルダイアログボックスをセットアップし，***TDialog*** : : ***SetupWindow*** を呼び出します。エディットコントロールに入力できる文字数を MAXPATH に制限します。さらに，***UpdateListBoxes*** と ***SelectFileName*** を呼び出します。

関連項目：***TDialog*** : : ***SetupWindow***, ***TFileDialog*** : : ***UpdateListBoxes***, ***TFileDialog*** : : ***SelectFileName***

UpdateFileName void UpdateFileName () ;

エディットコントロールのテキストに ***PathName*** を設定し，テキストを選択します。

UpdateListBoxes BOOL UpdateListBoxes () ;

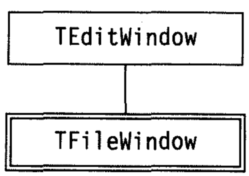
ファイルをオープンする際に，ファイルリストボックスとディレクトリリストボックスの更新を試みます。セーブする際には，ディレクトリリストボックスの更新を試みます。更新されると，***UpdateFileName*** を呼び出して真を返します。更新されなかった場合は偽を返します。

関連項目：***TFileDialog*** :: ***UpdateFileName***

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。

TFileWindow　　FILEWND.H



TEditWindow から派生したファイル編集用のウィンドウです。File ウィンドウはファイルダイアログボックスを使ってファイルのオープンとセーブをします。***TFileWindow*** のデータメンバとメンバ関数は，ファイルダイアログボックスを管理し，代表的なファイルコマンド Open, Read, Write, Save, SaveAs に対する自動応答を管理します。

データメンバ

FileName LPSTR FileName ;

編集対象のファイルの名前が入ります。ファイルダイアログボックスで使います。セーブ用関数と読み込み用関数でも使われます。

IsNewFile BOOL IsNewFile ;

ユーザが現在編集しているファイルが新しいファイルであれば真になり，以前にオープンされたファイルであれば偽になります。

メンバ関数

constructor TFileWindow (PTWindowsObject AParent, LPSTR ATitle, LPSTR AFileName, PTModule AModule = NULL) ;

TEditWindow のコンストラクタを呼び出して *AParent* と *ATitle* を渡します。***FileName*** に *AFileName* のコピーを，***IsNewFile*** に真を設定します。

constructor TFileWindow (StreamableInit) ; protected

TFileWindow のストリームコンストラクタです。***TEditWindow*** のストリームコンストラクタを呼び出します。指定されたストリームのデータで ***TFileWindow*** のインスタンスを構築するときに呼び出されます。

desutructor virtual ~TFileWindow () ;

TFileWindow 名を保持するために確保されたメモリを解放します。

build static PTStreamable build () ;

TFileWindow (***StreamableInit***)コンストラクタを呼び出します。型 ***TFileWindow*** のオブジェクトを構築してから，データメンバがストリームから読み込まれます。

CanClear virtual BOOL CanClear () ;

Editor のテキストを消去していいかどうかを示す論理値を返します。テキストに変更がなければ真を返します。テキストに変更があれば，ユーザにファイルをセーブするか，テキストを消去するかの判断を要求します。

CanClose virtual BOOL CanClose () ;

TFileWindow にクローズを許可する場合は真を返します。***CanClear*** の呼び出しの結果を返します。

CMFileNew virtual void CMFileNew (RTMessage Msg)
= [CM_FIRST + CM_FILENEW] ; protected

CM_FILENEW コマンド識別子のついた File New コマンドを受け取ると，***NewFile*** を呼び出します。

CMFileOpen virtual void CMFileOpen (RTMessage Msg)
= [CM_FIRST + CM_FILEOPEN] ; protected

CM_FILEOPEN コマンド識別子のついた File Open コマンドを受け取ると，***Open*** を呼び出します。

CMFileSave virtual void CMFileSave (RTMessage Msg)
= [CM_FIRST + CM_FILESAVE] ; protected

CM_FILESAVE コマンド識別子のついた File Save コマンドを受け取ると，***Save*** を呼び出します。

CMFileSaveAs virtual void CMFileSaveAs (RTMessage Msg)
= [CM_FIRST + CM_FILESAVEAS] ; protected

CM_FILESAVEAS コマンド識別子のついた File SaveAs コマンドを受け取ると，***SaveAs*** を呼び出します。

NewFile void NewFile () ;

CanClear を呼び出して，***Editor*** のテキストを消去してもよいことが確認されたら，新しいファイルの編集を始めます。

Open void Open () ;

Editor のテキストを消去してもよいことが確認されたら，新しいファイルをオープンします。*CanClear* を呼び出して，真が返されればファイルダイアログボックスを表示し，ユーザから新しいファイル名を取得します。新しいファイル名は，*ReplaceWith* を呼び出して渡します。

Read BOOL Read () ;

前回指定したファイルの内容を *Editor* に読み込みます。*IsNewFile* に偽を設定します。読み込み操作が成功すれば真を返します。

read virtual Pvoid read'(Ripstream is) ; protected

TEditWindow : : *read* を呼び出して基本クラス *TEditWindow* のオブジェクトを読み込みます。さらに，*FileName* を読み込みます。*FileName* がヌルの場合は *IsNewFile* に真を設定します。ヌルでなければ偽を設定します。

ReplaceWith void ReplaceWith (LPSTR AFileName) ;

SetFileName と *Read* を呼び出し，現在編集中のファイルを指定された名前のファイルと置き換えます。

Save BOOL Save () ;

Editor の内容に加わった変更をファイルにセーブします。ただし，*Editor* －＞*IsModified* が偽を返した場合は，単に真を返し，変更がないかまたは変更があっても既にセーブ済みであることを示します。それ以外は，*IsNewFile* が真であれば *SaveAs* を呼び出し，*IsNewFile* が偽であれば *Write* を呼び出します。どちらかの呼び出しの結果を返します。

SaveAs BOOL SaveAs () ;

File Save ダイアログボックスの実行により，ユーザが指定した名前のファイルに *Editor* の内容をセーブします。ユーザが OK を選択すれば，*SetFileName* と *Write* を呼び出します。ファイルがセーブされれば真を返します。

SetFileName void SetFileName (LPSTR AFileName) ;

FileName を設定してウィンドウのタイトルを更新します。

SetupWindow　virtual void SetupWindow () ;　　protected

TEditWindow :: ***SetupWindow*** を呼び出して，エディットウィンドウの ***Editor*** エディットコントロールを作成します。ウィンドウのタイトルに ***FileName*** を設定します。ファイル名の指定がなければタイトルなしになります。

Write　BOOL Write () ;

FileName で指定した名前のファイルに ***Editor*** の内容をセーブします。書き込み操作が成功すれば真を返します。

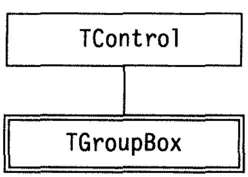
write　virtual void write (Ropstream os) ;　　protected

TEditWindow :: ***write*** を呼び出し，基本クラス ***TEditWindow*** のオブジェクトを書き込みます。さらに，***FileName*** を書き込みます。書き込み操作が成功すれば真を返します。

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。

TGroupBox　　GROUPBOX.H



この関数のインスタンスは，Windows の対応するグループボックス要素を表わすインターフェースオブジェクトです。ダイアログボックスやダイアログウィンドウ（***TDialog***）に使うことは通常ありませんが，ウィンドウのグループボックスを作成するのに使います。

画面上のグループボックスの役割は，チェックボックスとラジオボタンの選択ボックスやその他のコントロールを表示上まとめて見やすくするだけですが，内

部では選択ボックスの状態を管理するという大切な役割を果しています。

たとえば，選択変更に対する応答方法を，グループ内のラジオボタンで統一したい場合があります。***TGroupBox*** から派生させたクラスで ***SelectionChanged*** メンバ関数を定義し直せば，これが可能になります。

グループ内のラジオボタンの選択変更に応答するには，グループボックスの親に応答を定義するという別の方法もあります。この方法では，グループボックスの ID を用いて，子 ID 応答メンバ関数を定義します。そうすると，ラジオボタンに選択状態の変更があり次第，グループボックスは子 ID メッセージを自動的に親に送ります。

データメンバ

NotifyParent BOOL NotifyParent ;

グループボックスの選択ボックスの状態変化を親に通知するかどうかを指示するフラグです。デフォルトは真です。

メンバ関数

constructor TGroupBox (PTWindowsObject AParent, int AnId, LPSTR AText, int X, int Y, int W, int H, PTModule AModule = NULL) ;

渡された親ウィンドウ（*AParent*），コントロール ID（*AnID*），対応テキスト（*AText*），親ウィンドウのクライアント領域の原点を基準とする座標（*X, Y*），幅（*W*），高さ（*H*）でグループボックスオブジェクトを構築します。同じような引数を使う ***TControl*** を呼び出し，***Attr.Style*** に ***BS_GROUPBOX*** を追加して ***WS_TABSTOP*** を除去します。***NotifyParent*** は真にセットされます。デフォルトではグループボックスのコントロールに生じる選択変化をグループボックスの親に知らせます。

関連項目：***TControl*** :: ***TControl***

constructor TGroupBox (PTWindowObject AParent, int ResourceId,
PTModule AModule = NULL) ;

TDialog のグループボックスコントロールに対応する ***TGroupBox*** オブジェクトを構築します。同じ引数を持つ ***TControl*** コンストラクタを呼び出します。グループボックスは転送データを持たないので，さらに ***DisableTransfer*** を呼び出してグループボックスを転送機構から除外します。

ResourceId 引数は，定義するグループボックスリソースに必ず対応させます。

関連項目：***TControl*** : : ***TControl, TWindowsObject*** : : ***DisableTransfer***

constructor TGroupBox (StreamableInit) ; protected

TGroupBox ストリームコンストラクタ。***TControl*** のストリームコンストラクタを呼び出します。入力ストリームのデータで ***TGroupBox*** のインスタンスを構築する場合に呼び出されます。

関連項目：***TControl*** : : ***TControl***

build static PTStreamable build () ;

TGroupBox(***StreamableInit***)コンストラクタを呼び出します。***TGroupBox*** 型のオブジェクトを構築した後に，データメンバをストリームから読み込みます。

GetClassName virtual LPSTR GetClassName () ; protected

TGroupBox の Windows 登録クラス名である「BUTTON」を返します。

read virtual Pvoid read (Ripstream is) ; protected

TWindow : : ***read*** を呼び出して ***TGroupBox*** 基本オブジェクトを読み込みます。さらに ***NotifyParent*** を読み込みます。

SelectionChanged virtual void SelectionChanged (int ControlId) ;

NotifyParent が真であれば，子ウィンドウ ID メッセージをグループボックスの親ウィンドウに送り，グループボックスの選択項目の変更を知らせます。このメンバ関数を定義し直して，グループ内のコントロールに起きた選択の変更をグループボックスに処理させることができます。

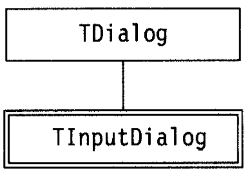
write　virtual void write (Ropstream os) ;　　protected

TWindow* : : *write を呼び出して ***TGroupBox*** 基本オブジェクトを書き出します。さらに ***NotifyParent*** を書き出します。

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。

TInputDialog　INPUTDIA.H



ユーザが入力したテキストを取得する汎用ダイアログボックスを提供します。入力ダイアログボックスを構築するとき，そのタイトル，プロンプト，デフォルトの入力テキストを指定します。

データメンバ

Buffer　LPSTR Buffer ;

ユーザから取得したテキストを返すバッファのポインタです。入力ダイアログボックスのコンストラクタに渡されるとき，エディットコントロールに初めに表示されるデフォルトのテキストが入ります。

BufferSize　WORD BufferSize ;

ユーザ入力を返すバッファのサイズが入ります。

Prompt　LPSTR Prompt ;

入力ダイアログボックスのプロンプトを指します。

メンバ関数

constructor　TInputDialog (PTWindowObject AParent ; LPSTR ATitle,
LPSTR APrompt, LPSTR ABuffer, WORD ABufferSize,
PTModule AModule = NULL) ;

TDialog のコンストラクタを呼び出し，*AParent*，リソース識別子，***SD_INPUTDIALOG***，*AModule* を渡します。ダイアログボックスのタイトルに *ATitle* を，プロンプトのスタティックコントロールに *APrompt* を設定します。データメンバ **Buffer, BufferSize** に *ABuffer, ABufferSize* を設定します。

関連項目：***TDialog*** : : ***TDialog***

constructor　TInputDialog (StreamableInit) ;　protected

TInputDialog のストリームコンストラクタです。***TDialog*** のストリームコンストラクタを呼び出します。入力ストリームのデータで ***TInputDialog*** のインスタンスを構築する場合に呼び出されます。

関連項目：***TDialog*** : : ***TDialog***

build　static PTStreamable build () ;

TInputDialog (***StreamableInit***)コンストラクタを呼び出します。***TInputDialog*** 型のオブジェクトを構築した後にデータメンバがストリームから読み込まれます。

read　virtual Pvoid read (Ripstream is) ;　protected

TDialog : : ***read*** を呼び出して，基本クラス ***TDialog*** のオブジェクトを読み込ませます。その後で，***Prompt*** を読み込みます。

SetupWindow　virtual void SetupWindow () ;　protected

ウィンドウをセットアップするときに，***TDialog*** : : ***SetupWindow*** を呼び出

します。さらに，ユーザが入力できる文字数を ***BufferSize*** より 1 少ない数に制限します。

TransferData void TransferData (WORD Direction) ;

入力ダイアログボックスのデータを転送します。*Direction* が ***TF_SETDATA*** であれば，ダイアログボックスのスタティックコントロールとエディットコントロールのテキストとして，***Prompt*** と ***Buffer*** のテキストを設定します。*Direction* が ***TF_GETDATA*** であれば，***Editor*** の現在のテキストを ***Buffer*** に書き込みます。

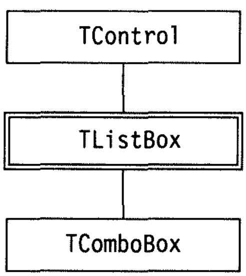
write virtual void write (Ropstream os) ; protected

TDialog : : ***write*** を呼び出し，基本クラス ***TDialog*** のオブジェクトを書き込ませます。その後で，***Prompt*** を書き込みます。

関連関数

operator<<と operator>>については 13 ページを参照してください。

TListBox LISTBOX.H



Windows のリストボックス要素に対応するインターフェースオブジェクトです。親の ***TWindow*** でリストボックスコントロールを作成するときは ***TListBox*** を使わなければなりません。***TListBox*** を使うとアプリケーションと ***TDialog*** のリストボックスコントロール間の通信が簡単になります。***TListBox*** のメンバ関数は派生クラス ***TComboBox*** のインスタンスにも使われます。

メンバ関数

constructor TListBox (PTWindowsObject AParent, int AnId, int X, int Y, int W, int H,
PTModule AModule = NULL) ;

渡された親ウィンドウ(*AParent*), コントロールID(*AnId*), 原点位置を親ウィンドウのクライアント領域とする座標(*X*, *Y*), 幅(*W*), 高さ(*H*)でリストボックスオブジェクトを構築します。***TControl*** コンストラクタを呼び出し, リストボックスのデフォルトスタイルとして ***LBS_STANDARD*** を追加します。その結果リストボックスには次のものが加わります。

- ***WS_BORDER*** による枠
- ***WS_VSCROLL*** による垂直スクロールバー
- ***LBS_SORT*** によるリスト項目のアルファベット順自動ソート
- ***LBS_NOTIFY*** による選択時の親ウィンドウへの通知

constructor TListBox (PTWindowsObject AParent, int ResourceId,
PTModule AModule = NULL) ;

TDialog のリストボックスコントロールに対応する ***TListBox*** オブジェクトを構築します。同様の引数で ***TControl*** のコンストラクタを呼び出します。引数 *ResourceId* は, 定義するリストボックスリソースに必ず対応させせます。

関連項目：***TControl :: TControl***

constructor TListBox (StreamableInit) ; protected

TListBox のストリームコンストラクタです。***TControl*** のストリームコンストラクタを呼び出します。入力ストリームのデータで ***TListBox*** のインスタンスを構築する場合に呼び出されます。

関連項目：***TControl :: TControl***

Addstring int AddString (LPSTR AString) ;

リストボックスに *AString* を追加し, リスト内の該当位置（先頭の位置は 0)を返します。エラーの場合は負の値を返します。リスト項目は自動的にソートされます。ただし, リスト項目の作成前にリストボックスオブジェクトのデータメンバ ***Attr.Style*** から ***LBS_SORT*** が除去された場合を除きます。

build static PTStreamable build () ;

TListBox (***StreamableInit***)コンストラクタを呼び出します。***TListBox*** 型のオブジェクトを構築した後に，データメンバをストリームから読み込みます。

ClearList void ClearList () ;

リスト内すべての項目を消去します。

DeleteString int DeleteString (int Index) ;

Index で渡された位置（0 以上）の項目をリストから削除し，残っているリスト項目数を返します。エラーの場合は負の値を返します。

FindExactString int FindExactString (LPSTR AString, int SearchIndex) ;

SearchIndex で渡された行番号からリストボックスの検索を開始し，文字列 *AString* とまったく同じ文字列をリストボックスから探します。最後の文字列まで照合しても該当する文字列がなければ，リストの先頭に戻って検索を始めます。該当文字列が見つかるまで，またはリスト全体を探し終わるまで検索を続けます。*SearchIndex* に −1 を渡すと，リストの先頭から検索します。成功すると，最初の該当文字列のインデックスを返します。エラーが起こると負の値を返します。

FindString int FindString (LPSTR AString, int SearchIndex) ;

上の ***FindExactString*** と同様に検索しますが，*AString* で始まる最初のエントリを探す点が異なります。

GetClassName virtual LPSTR GetClassName () ; protected

TListBox の Windows 登録クラス名である「LISTBOX」を返します。

GetCount int GetCount () ;

リストボックス内の項目数を返します。エラーの場合は負の値を返します。

GetMsgID virtual WORD GetMsgID (WORD AMsg) ; protected

指定された ObjectWindows メッセージ識別子に対応する Windows リストボックスメッセージの識別子を返します。***TComboBox*** とその派生クラスが ***TListBox*** のメンバ関数を多数継承できるようにします。

GetSelCount int GetSelCount () ;

リストボックス内で選択されている項目の数を返します。単一項目選択式リストボックス，複数項目選択式リストボックス，コンボボックスに用います。

GetSelIndex int GetSelIndex () ;

現在の選択されている項目の負でないインデックス（0 以上）を返します。選択中の項目がなければ負の値を返します。単一項目選択式のリストボックスに用います。

GetSelIndexes int GetSelIndexes (Pint Indexes, int MaxCount) ;

複数項目選択式のリストボックスに用います。*Indexes* 配列に最大で *MaxCount* までの選択されている文字列のインデックスを挿入します。*Indexes* に書き込まれた項目数を返します。単一項目選択式リストボックスとコンボボックスの場合は -1 を返します。

GetSelString int GetSelString (LPSTR AString, int MaxChars) ;

現在選択されている項目の長さが *MaxChars* 以下であればこの項目を取得し，最後のヌルも含めて *AString* で返します。戻り値は文字列の長さです。エラーの場合は負の値を返します。選択されている文字列がなければ，-1 を返します。単一項目選択式のリストボックスに用います。

GetSelStrings int GetSelStrings (LPSTR *Strings, int MaxCount, int MaxChars) ;

複数項目選択式のリストボックスに用います。現在選択されている項目を取得し，*MaxCount* までの文字列を *Strings* に書き込みます。*Strings* 配列の各エントリには，*MaxChars* の文字数と末尾のヌルが入るスペースをとります。*Strings* に書き込まれた項目数を返します。単一項目選択式のリストボックスとコンボボックスの場合は -1 を返します。

GetString int GetString (LPSTR AString, int Index) ;

Index で渡された位置(0 以上）の項目を取得して *AString* で返します。戻り値は文字列の長さです。エラーの場合は負の値を返します。

GetStringLen int GetStringLen (int Index) ;

Index で渡されたインデックス位置の項目の（末尾のヌルを除く）文字列の長さを返します。エラーの場合は負の値を返します。

InsertString int InsertString (LPSTR AString, int Index) ;

Index で渡されたリストボックスの位置に *AString* を挿入し，リスト内のその項目の実際位置（0 以上）を返します。エラーの場合は負の値を返します。リストは再ソートされません。*Index* が-1 であれば，文字列をリストの最後に追加します。

SetSelIndex int SetSelIndex (int Index) ;

Index で渡された位置（0 以上）のリスト項目を選択します。*Index* が -1 であればリストボックスの選択内容はすべてクリアされます。エラーの場合の戻り値は負の数です。単一項目選択リストボックスに用います。

SetSelIndexes int SetSelIndexes (Pint Indexes, int NumSelections, BOOL ShouldSet) ;

複数項目選択式のリストボックスに用います。*Indexes* 配列で指定されたインデックス位置にある文字列を対応するリストボックスから選択または選択解除します。*ShouldSet* が真であれば，インデックス位置の文字列が選択されてハイライト表示になります。*ShouldSet* が偽であれば，ハイライトが除去され，選択が解除されます。選択または選択解除が成功した文字列数を返します。単一項目選択式のリストボックスとコンボボックスの場合は-1 を返します。*NumSelections* が 0 未満の場合はすべての文字列が選択または選択解除されます。失敗の場合は負の値を返します。

SetSelString int SetSelString(LPSTR AString, int AIndex) ;

AString で渡されたテキストで始まる文字列の最初の項目を，*AIndex* で渡された位置（0 以上）を超える位置から選択します。*AIndex* が -1 であれば，リスト全体を最初から検索します。戻り値は新たに選択された項目の位置です。エラーの場合は負の値です。単一項目選択式のリストボックスに用います。

SetSelStrings int SetSelString(LPSTR *Prefixes, int NumSelections, BOOL ShouldSet) ;

複数項目選択式のリストボックスに用います。*Prefixes* 配列で指定されたプレフィクスで始まる文字列を対応リストボックスから選択します。該当文字列が見つかるか，リスト全体の検索が終わるまで，リストの先頭から各文字列を探します。*ShouldSet* が真であれば，該当文字列が選択されてハイライト表示になります。*ShouldSet* が偽であれば，該当文字列のハイライトが除去され，選択が解除されます。選択または選択解除が成功した文字列数を返します。単一項目選択式のリストボックスとコンボボックスの場合は-1 を返します。*NumSelections* が 0 未満の場合は，すべての文字列が選択または選択解除されます。失敗の場合は負の値を返します。

Transfer virtual WORD Transfer (Pvoid DataPtr, WORD TransferFlag) ;

転送バッファとの間でリストボックスの項目と選択内容を転送します。*TransferFlag* が ***TF_GETDATA*** であれば転送バッファから転送，***TF_SETDATA*** であれば転送バッファに転送します。*TransferBuffer* はリストボックスのデータ転送バッファとして用いる ***PTListBoxData*** オブジェクトを指すことになっています。

戻り値は ***PTListBoxData*** （構造体ではなくポインタ）の大きさです。データを転送せずに大きさを取得するには，*TransferFlag* として ***TF_SIZEDATA*** を渡します。

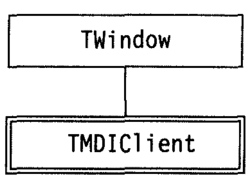
転送バッファでは，以上の構造体を指すポインタを必ず使います。転送バッファに構造体のコピーを埋め込むことも，構造体を転送バッファとして使うこともできません。

関連項目：***TListBoxData, TWindowsObject*** :: ***TransferBuffer***

関連関数

operator＜＜と operator＞＞については 13 ページを参照してください。

TMDIClient MDI.H



TMDIClient オブジェクトが表わすマルチドキュメントインターフェース（MDI）のクライアントウィンドウは，親の ***TMDIFrame*** の MDI 子ウィンドウを管理します。

データメンバ

ClientAttr LPCLIENTCREATESTRUCT ClientAttr ;

MDI クライアントウィンドウの属性の構造体のポインタを保持します。

メンバ関数

constructor TMDIClient (PTMDIFrame AParent, PTModule AModule = NULL) ;

MDI クライアントウィンドウオブジェクトを構築します。***TWindow*** のコンストラクタを呼び出して *AParent* と *AModule* を渡します。***Attr.Id*** にデフォルトのクライアントウィンドウ識別子である ***ID_MDICLIENT*** をセットします。***Attr.Style*** には ***WS_CHILD*** | ***WS_VISIBLE*** | ***WS_GROUP*** | ***WS_TABSTOP*** | ***WS_CLIPCHILDREN*** をセットします。*ClientAttr* を初期化して，そのメンバ ***idFirstChild*** を ***ID_FIRSTMDICHILD*** をセットします。

関連項目：***TWindow* :: *TWindow***

constructor TMDIClient (PTMDIFrame AParent, HWND AnHWindow,
PTModule AModule = NULL) ;

TWindow のコンストラクタを呼び出して，ハンドルが指定された MDI クライアントウィンドウに対応する MDI クライアントオブジェクトを構築します。*AHWindow* と *AModule* を渡します。親ウィンドウが ObjectWindows 以外のウィンドウである DLL で MDI クライアントウィンドウを構築できます。

constructor TMDIClient (StreamableInit) ; protected

TMDIClient のストリームコンストラクタです。***TWindow*** のストリームコンストラクタを呼び出します。入力ストリームのデータで ***TMDIClient*** のインスタンスを構築する場合に呼び出されます。

関連項目：***TWindow* :: *TWindow***

destructor ~TMDIClient () ;

ClientAttr 構造体があれば，それを解放します。

関連項目：***TWindow*** :: ~***TWindow***

ArrangeIcons virtual void ArrageIcons () ;

MDI クライアントウィンドウの下部に MDI 子ウィンドウを整列させます。

build static PTStreamable build () ;

TMDIClient (***StreamableInit***)コンストラクタを呼び出します。***TMDIClient*** 型のオブジェクトを構築した後に，データメンバをストリームから読み込みます。

CascadeChildren virtual void CascadeChildren () ;

アイコンになっていないすべての MDI 子ウィンドウのサイズと位置を調整して MDI クライアントウィンドウ内に配置します。各子ウィンドウはオーバーラップしますがタイトルバーは表示されます。

GetClassName virtual LPSTR GetClassName () ; protected

TMDIClient の Windows 登録クラス名である「MDICLIENT」を返します。

read virtual Pvoid read (Ripstream is) ; protected

TWidow :: ***read*** を呼び出して基本オブジェクト ***TWindow*** を読み込ませます。***ClientAttr*** -> ***hWindowMenu*** に 0 をセットし，***ClientAttr*** -> ***idFirstChild*** を読み込みます。

TileChildren virtual void TileChildren () ;

アイコンになっていないすべての MDI 子ウィンドウのサイズと位置を調整し，オーバーラップさせずに MDI クライアントウィンドウのクライアント領域全体に並べます。

WMMDIActivate virtual void WMMDIActivate(RTMessage) = [WM_FIRST + WM_MDIACTIVATE] ; protected

送られてきた ***WM_MDIACTIVATE*** メッセージに応答します。***TWindow*** :: ***WMMDIActivate*** をオーバーライドして，かわりに ***DefWndProc*** を呼び出します。

派生クラスで ***WMMDIActivate*** を定義し直す場合は，必ず基本クラスの ***WMMDIActivate*** を呼び出します。特別な理由がない限りは，他の処理をすべて終えてから呼び出します。

WMPaint virtual void WMPaint (RTMessage Msg) = [WM_FIRST + WM_PAINT] ; protected

TWindow :: ***WMPaint*** を定義し直して ***DefWndProc*** を呼び出します。

関連項目：***TWindow*** :: ***WMPaint***, ***TScroller*** :: ***BeginView***, ***TScroller*** :: ***EndView***

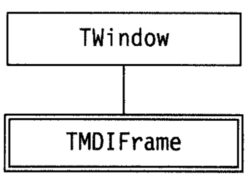
write virtual void write (Ropstream os) ; protected

TWindow :: ***write*** を呼び出して基本オブジェクト ***TWindow*** を書き込みます。さらに ***ClientAttr*** - > ***idFirstChild*** を書き込みます。

関連関数

operator＜＜と operator＞＞については 13 ページを参照してください。

TMDIFrame MDI.H



TMDIFrame が表わすマルチドキュメントインターフェース（MDI）のフレームウィンドウは，MDI 対応アプリケーションのメインウィンドウとして用いるオーバーラップウィンドウです。Windows の要求に従い，***TMDIClient*** オブジェクトが表わす MDI クライアント「ウィンドウ」の作成と初期化を自動的に処理します。***TMDIFrame*** はメンバ関数は ***TileChildren*** や ***CloseChildren*** など MDI 子ウィンドウを操作するメンバ関数も提供します。

データメンバ

ActivateChild PTWindow ActiveChild ;

TMDIFrame のアクティブな MDI 子ウィンドウを指すポインタです。***ActivateChild*** は子ウィンドウにより ***WMMDIActivate*** メッセージ応答メンバ関数で設定されます。***TMDIFrame*** のコンストラクタで初期化し，***read*** 関数で読み込み，***write*** 関数で書き込みます。

ChildMenuPos int ChildMenuPos ;

MDI ウィンドウの最上位メニューにおける子ウィンドウのサブメニューの位置が入ります。左上が 0 位置です。

ClientWnd PTMDIClient ClientWnd ;

TMDIClient のインスタンスである ***TMDIFrame*** のクライアントウィンドウを指すポインタです。

メンバ関数

constructor TMDIFrame (LPSTR ATitle, int MenuId,
PTModule AModule = NULL) ;

タイトル（*ATitle*）と整数リソース識別子（*MenuId*）で MDI フレームウィンドウを構築します。

constructor TMDIFrame (LPSTR ATitle, LPSTR MenuName,
PTModule AModule = NULL) ;

タイトル（*ATitle*）と文字列リソース識別子（*MenuName*）で MDI フレームウィンドウを構築します。(MDI フレームウィンドウにはメニューが必要です。親は持てません。) デフォルトでは ***ChildMenuPos*** に 0 をセットし，子ウィンドウメニューの位置が左上であることを示します。派生クラスのコンストラクタで，次のように ***ChildMenuPos*** を設定し直すことができます。

```
TMyMDIFrame : : TMyMDIFrame(LPSTR ATitle, LPSTR MenuName,
    PTModule AModule = NULL) : TMDIFrame(ATitle, AMenu,
    AModule)
 {
     ChildMenuPos = 3;
 }
```

constructor TMDIFrame (HWND AnHWindow, HWND ClientWnd) ;

ハンドルで指定された MDI ウィンドウに対応する MDI ウィンドウオブジェクトを構築します。***TWindow*** コンストラクタを呼び出した後で，ハンドルで指定されたクライアントウィンドウに対応する MDI クライアントオブジェクトを構築します。

constructor TMDIFrame (StreamableInit) ; protected

TMDIFrame のストリームコンストラクタです。***TWindow*** のストリームコンストラクタを呼び出します。入力ストリームのデータで ***TMDIFrame*** のインスタンスを構築するときに呼び出されます。

関連項目：***TWindow*** : : ***TWindow***

destructor virtual ~TMDIFrame () ;

MDI クライアントウィンドウオブジェクトを削除します。

関連項目：***TWindow*** : : ***TWindow***

ArrangeIcons virtual void ArrageIcons () ;

クライアントウィンドウのメンバ関数 ***ArrangeIcons*** を呼び出して，MDI クライアントウィンドウの下部に，アイコンになっている MDI 子ウィンドウを横一列に並べます。

関連項目：***TMDIClient*** : : ***ArrangeIcons***

build static PTStreamable build () ;

TMDIFrame (***StreamableInit***)コンストラクタを呼び出します。***TMDIFrame*** 型のオブジェクトを構築した後でそのデータメンバでストリームから読み込まれます。

CascadeChildren virtual void CascadeChildren () ;

クライアントウィンドウのメンバ関数 ***CascadeChildren*** を呼び出して，アイコンになっていないすべての MDI 子ウィンドウのサイズと位置を調整し，MDI クライアントウィンドウ内に収まるように配置します。

関連項目：***TMDIClient*** : : ***CascadeChildren***

CloseChildren virtual BOOL CloseChildren () ;

すべての MDI 子ウィンドウをクローズできる状態であればすべてクローズします。すべての子がクローズされるか，または子ウィンドウが 1 つもなければ，真を返します。クローズできない子が 1 つでもあれば，偽を返します。

関連項目：***TWindow*** : : ***CanClose***

CMArrangeIcons virtual void CMArrangeIcons (RTMessage Msg)
= [CM_FIRST + CM_ARRANGEICONS] ; protected

ID が ***CM_ARRANGEICONS*** のメニュー選択に応答し，***ArrangeIcons*** を呼び出します。

関連項目：***TMDIFrame*** : : ***ArrangeIcons***

CMCascadeChildren virtual void CMCascadeChildren (RTMessage Msg)
= [CM_FIRST + CM_CASCADECHILDREN] ; protected

ID が ***CM_CASCADECHILDREN*** のメニュー選択に応答し，***CascadeChildren*** を呼び出します。

関連項目：***TMDIFrame*** : : ***CascadeChildren***

CMCloseChildren virtual void CMCloseChildren (RTMessage Msg)
= [CM_FIRST + CM_CLOSECHILDREN] ; protected

ID が ***CM_CLOSECHILDREN*** のメニュー選択に応答し，***CloseChildren*** を呼び出します。

関連項目：***TMDIFrame*** : : ***CloseChildren***

CMCreateChild virtual void CMCreateChild (RTMessage Msg)
= [CM_FIRST + CM_CREATECHILD] ; protected

メニューIDが ***CM_CREATECHILD*** のメニューが選択されると，***CreateChild*** を呼び出して子ウィンドウを新たに作成します。

関連項目：***TMDIFrame*** : : ***CreateChild***

CMTileChildren virtual void CMTitleChildren (RTMessage Msg)
= [CM_FIRST + CM_TILECHILDREN] ; protected

IDが ***CM_TILECHILDREN*** のメニュー選択に応答し，***TileChildren*** を呼び出します。

関連項目：***TMDIFrame*** : : ***TileChildren***

CreateChild virtual PTWindowsObject CreateChild () ;

InitChild と ***MakeWindow*** を呼び出して MDI 子ウィンドウの構築と作成を行ないます。この新しい MDI 子ウィンドウのポインタを返します。

関連項目：***TMDIFrame*** : : ***InitChild, TModule*** : : ***MakeWindow***

TM
Class

GetClassName virtual LPSTR GetClassName () ; protected

TMDIWindow の Windows 登録クラス名である「OWLMDIFrame」を返します。

GetClient virtual PTMDIClient GetClient () ;

ClientWnd に格納されている MDI クライアントウィンドウのポインタを返します。

GetWindowClass virtual void GetWindowClass (WNDCLASS _FAR & AWndClass) ; protected

デフォルトのウィンドウクラス構造体を変更して ***AWndClass*** で戻します。***TWindow*** : : ***GetWindowClass*** を呼び出し，***AWndClass.Style*** に 0 をセットします。

関連項目：***TWindow*** : : ***GetWindowClass***

InitChild virtual PTWindowObject InitChild () ;

TWindow のインスタンスを MDI 子ウィンドウとして構築し，そのポインタを返します。派生させた MDI ウィンドウクラスでこのメンバ関数を定義し直すと，次の例のように派生させた MDI 子クラスのインスタンスを構築できます。

```
PTWindowsObject TMyMDIFrame : : InitChild ()
 {
     return new TMyMDIChild(this, "") ;
 }
```

関連項目：***TMDIFrame*** : : ***CreateChild***

InitClientWindow virtual void InitClientWindow () ;

MDI クライアントウィンドウを ***TMDIClient*** のインスタンスとして構築し，そのポインタを ***ClientWnd*** に格納します。

read virtual Pvoid read (Ripsream is) ; protected

TWindow : : read を呼び出して，基本クラス ***TWindow*** のオブジェクトを読み込ませます。その後で ***ClientWnd*** と ***ChildMenuPos*** を読み込みます。

SetupWindow virtual void SetupWindow () ; protected

InitClientWindow を呼び出して，MDI クライアントウィンドウを構築します。子リストから MDI クライアントウィンドウを除去し，別の処理に回します。派生クラスで ***SetupWindow*** を定義し直す場合は，***TMDIFrame*** : : ***SetupWindow*** を必ず明示的に呼び出してください。

関連項目：***TMDIFrame*** : : ***InitClientWindow***

TileChildren virtual void TileChildren () ;

クライアントウィンドウのメンバ関数 ***TileChildren*** を呼び出して，アイコンになっていないすべての MDI 子ウィンドウのサイズと位置を調整し，MDI クライアントウィンドウに収まるように並べます。

関連項目：***TMDIClient*** : : ***TileChildren***

WMActivate virtual void WMActivate (RTMessage Msg) = [WM_FIRST + WM_ACTIVATE] ; protected

送られてきた ***WM_ACTIVATE*** メッセージに応答します。MDI 子ウィンドウのフレームウィンドウがアクティブ化または非アクティブ化されても，MDI 子ウィンドウは ***WM_MDIACTIVATE*** メッセージを受け取らないので，アク

ティブな子ウィンドウのメンバ関数 ***ActivationResponse*** をここで呼び出します。

関連項目：***TWindow*** :: ***ActivationResponse,***
TWindowsObject :: ***WMActivate***

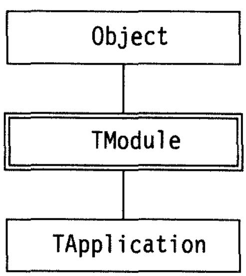
write virtual void write (Ropstream os) ; protected

TWindow :: ***write*** を呼び出して，基本クラス ***TWindow*** のオブジェクトを書き込ませます。その後で ***ClientWnd*** と ***ChildMenuPos*** を書き込みます。

関連関数

operator＜＜と operator＞＞については 13 ページを参照してください。

TModule MODULE.H



ObjectWindows のダイナミックリンクライブラリ(DLL)が構築する ***TModule*** のインスタンスは，ライブラリ(DLL)モジュールの「代役」として動作します。ObjectWindows アプリケーションは，***TModule*** から派生した ***TApplication*** のインスタンスを構築します。***TModule*** はライブラリモジュールとアプリケーションモジュールの両方に共通する動作を定義します。***TModule*** のメンバ関数でウィンドウのメモリ管理とエラー処理をサポートしています。

データメンバ

hinstance HANDLE hInstance ;

Windows アプリケーションまたは DLL モジュールの実行用インスタンスのハンドルが入ります。リソースをロードするときは，このハンドルを引数として Windows に渡さなければなりません。

lpCmLine LPSTR lpCmdLine ;

モジュールをロードしたときに渡されたコマンドラインのコピーを指します。ヌルで終わる文字列です。

Name LPSTR Name ;

アプリケーションまたは DLL モジュールの名前を保持します。モジュールのコードで内部的に使用できます。

Status int Status ;

モジュールの状態が入ります。アプリケーションモジュールのメインウィンドウを作成したときに発生したエラーを ***WinMain*** の戻り値として示すように ObjectWindows で設定できます。あるいは，***LibMain*** が返す値を判定するように設定すれば，DLL ライブラリモジュールを起動したときに発生したエラーを示すことができます。

メンバ関数

constructor TModule (LPSRT AName, HANDLE AnInstance, LPSTR ACmdLine) ;

指定された引数でデータメンバを初期化します。*ACmdLine* のコピーを作成し，割り当てられた文字列を ***lpCmdLine*** に設定します。エラーが発生すると ***Status*** に ***EM_INVALIDMODULE*** を設定します。

destructor virtual ~TModule () ;

TModule オブジェクトのデストラクタです。***lpCmdLine*** を削除します。

Error virtual void Error (int ErrorCode) ;

ErrorCode で渡されたエラー値に応じてエラーを処理します。*ErrorCode* は独自に定義したエラーか ***EM*** _定数とします。***EM*** _定数については第3章『**ObjectWindows のその他の構成要素**』を参照してください。

Error はメッセージボックスにエラーコードを表示し，そのまま処理を続行するかどうかをユーザに問い合わせます。続行しない場合はプログラムの実行を中断します（プログラムの終了）。続行した場合はプログラムが回復するという保証はありません。

ExecDialog virtual int ExecDialog (PTWindowsObject ADialog) ;

指定されたモードつきダイアログボックスオブジェクトが有効であれば，そのダイアログボックスを実行します。***ValidWindow*** を呼び出して，オブジェクトが有効であるか判定します。有効であれば，ダイアログボックスのメンバ関数 ***Execute*** を呼び出し，その結果を返します。無効であれば，***IDCANCEL*** を返します。無効オブジェクトと実行できなかったオブジェクト(***Status*** がゼロ以外)は，***ExecDialog*** によって削除されます。これ以外のオブジェクトは ***Execute*** によって削除されます。

関連項目：***TModule*** :: ***ValidWindow, TDialog*** :: ***Execute, TModule*** :: ***MakeWindow***

GetClientHandle HWND GetClientHandle (HWND AnHWindow) ;

指定されたハンドルのウィンドウにある MDI クライアントウィンドウのハンドルを返します。ウィンドウに MDI クライアントウィンドウがなければ 0 を返します。

GetParentObject virtual PTWindowsObject GetParentObject (HWND ParentHandle) ;

DLL 内で呼び出された場合は，親ウィンドウオブジェクトのポインタを取得します。ObjectWindows アプリケーションから呼び出されると，このアプリケーションの親ウィンドウオブジェクトのポインタを返します。ObjectWindows 以外から呼び出されると，呼び出し側アプリケーションの親ウィンドウに対し DLL のオブジェクトとして機能するように構築されたオブジェクトのポインタを返します。

hashValue virtual hashValueType hasValue () const ;

コンテナクラスで必要なメンバ関数です。***Object*** クラスの純粋仮想関数を定義し直します。***TModule*** のハッシュ値 ***hInstanse*** を返します。

isA virtual classType isA () const ;

コンテナクラスで必要なメンバ関数です。***Object*** クラスの純粋仮想関数を定義し直します。***TModule*** のクラス識別子 ***moduleClass*** を返します。

IsEqual virtual int isEqual (RCObject module) const ;

コンテナクラスで必要なメンバ関数です。***Object*** クラスの純粋仮想関数を定義し直します。this と *module* の ***hInstance*** が等しければ真を返します。

LowMemory BOOL LowMemory () ;

セーフティプールを使いきると真を返します。

関連項目：***TModule*** :: ***RestoreMemory***

MakeWindow virtual PTWindowsObject MakeWindow (PTWindowsObject AWindowsObject) ;

オブジェクトが有効であれば，指定されたオブジェクトに対応するウィンドウまたはモードなしダイアログボックスを作成します。***ValidWindow*** を呼び出してオブジェクトが有効であるか判定させます。有効であれば，オブジェクトのメンバ関数 ***Create*** を呼び出します。***Create*** が真を返せば，***AWindowsObject*** を返します。

➡ 無効オブジェクトと作成できなかったオブジェクトは ***MakeWindow*** が値を返す前に削除され，NULL が返ります。

関連項目：***TModule*** :: ***ValidWindow, TWindow*** :: ***Create***

nameOf virtual Pchar nameOf () const ;

コンテナクラスで必要なメンバ関数です。***Object*** クラスの純粋仮想関数を定義し直します。***TModule*** のクラス識別子文字列「TModule」を返します。

printOn　virtual void printOn (Rostream outputStream)　const ;

コンテナクラスで必要なメンバ関数です。***Object*** クラスの純粋仮想関数を定義し直します。モジュールのインスタンスハンドル(***hInstance***)を 16 進表示で，指定されたストリームに出力します。

RestoreMemory　void RestoreMemory () ;

セーフティプールが確保されていなければ，確保します。

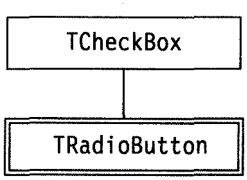
関連項目：***SafetyPool*** : : ***Allocate, TModule*** : : ***LowMemory***

ValidWindow　virtual PTWindowsObject ValidWindow (PTWindowsObject AWindowsObject) ;

AWindowsObject が有効なオブジェクトかどうかを判定します。*AWindowsObject* が有効であれば，このオブジェクトのポインタを返します。無効であれば，オブジェクトを削除して NULL を返します。***AWindowsObject*** は次の場合に無効になります。オブジェクトを割り当てるとセーフティプールが限界になるため ***LowMemory*** が真を返した場合，または *AWindowsObject* のデータメンバ ***Status*** がゼロ以外の場合です。

関連項目：***TModule*** : : ***LowMemory, TWindowObject*** : : ***Status***

TRadioButton　　　　RADIOBUT.H



Windows の対応するラジオボタン要素を表わすインターフェースオブジェクトです。親の ***TWindow*** でラジオボタンコントロールを作成するときに必ず使ってください。また，***TRadioButton*** を使うとアプリケーションと ***TDialog*** のラジオボタンコントロール間の通信が簡単になります。

ラジオボタンにはチェック状態と非チェック状態の2つの状態があります。その状態管理用メンバ関数は基本クラス ***TCheckBox*** から継承しています。コントロールを形状と機能別にまとめるグループ（***TGroupBox***）にラジオボタンを所属させることもできます。

メンバ関数

constructor TRadioButton (PTWindowsObject AParent, int AnId, LPSTR ATitle,
int X, int Y, int W, int H, PTGroupBox AGroup,
PTModule AModule = NULL) ;

渡された親ウィンドウ（*AParent*），コントロールID（*AnId*），対応テキスト（*ATitle*），親ウィンドウのクライアント領域の原点を基準とする座標（*X*，*Y*），幅（*W*），高さ（*H*），対応グループボックス（*AGroup*）でラジオボタンオブジェクトを構築します。同様の引数を使う ***TCheckBox*** を呼び出します。さらに ***Attr.Style*** データメンバに ***WS_CHILD*** | ***WS_VISIBLE*** | ***BS_RADIOBUTTON*** をセットします。

関連項目：***TControl*** :: ***TControl***

constructor TRadioButton (PTWindowsObject AParent, intResourceId,
PTGroupBox AGoup, PTModule AModule = NULL) ;

TDialog のラジオボタンコントロールに対応する ***TRadioButton*** オブジェクトを構築します。同じ引数を持つ ***TCheckBox*** コンストラクタを呼び出します。引数 ***ResourceId*** は，定義するラジオボタンリソースに必ず対応させてください。

constructor TRadioButton (StreamableInit) ; protected

TRadioButton のストリームコンストラクタです。***TCheckBox*** のストリームコンストラクタを呼び出します。入力ストリームのデータで ***TRadioButton*** のインスタンスを構築する場合に呼び出されます。

関連項目：***TCheckBox*** :: ***TCheckBox***

BNClicked　virtual void BNClicked (RTMessage Msg) = [NF_FIRST + BN_CLICKED] ; protected

Windows メッセージで ***BN_CLICKED*** が送られてくると応答します。

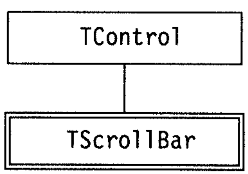
build　static PTStreamable build () ;

TRadioButton (***StreamableInit***)コンストラクタを呼び出します。***TRadioButton*** 型のオブジェクトを構築した後に，データメンバがストリームから読み込まれます。

関連関数

operator＜＜と operator＞＞については 13 ページを参照してください。

TScrollBar　　SCROLLBA.H



このオブジェクトはスタンドアロンの垂直スクロールバーコントロールと水平スクロールバーコントロールを表わします。大部分のメンバ関数は，スクロールボックスの位置と範囲を管理します。

TScrollBar 型の特徴は，Windows のスクロールバーメッセージに応じてスクロールボックス位置を自動的に調節する一連の通知別メンバ関数です。

☞ ***TScrollBar*** オブジェクトは ***WS_HSCROLL*** または ***WS_VSCROLL*** スタイルを属性として持つウィンドウでは使えません。

データメンバ

LineMagnitude int LineMagnitude ;

スクロールバー矢印のクリックによる小刻みなスクロール単位数を表わします。***TScrollBar*** のコンストラクタは，デフォルトで ***LineMagnitude*** に 1 をセットします。スクロール範囲は，デフォルトで 0 から 100 です。

関連項目：***TScrollBar*** : : ***SetupWindow***

PageMagnitude int PageMagnitude ;

ユーザがスクロールバーのスクロール領域をクリックして大幅なスクロールを要求したときにスクロールバーをスクロールさせる単位数です。***TScrollBar*** のコンストラクタはデフォルトで ***PageMagnitude*** に 10 をセットします。スクロール範囲はデフォルトで 0 から 100 です。

メンバ関数

constructor TScrollBar (PTWindowsObject AParent, int AnId, int X, int Y, int W, int H, BOOL IsHScrollBar, PTModule AModule = NULL) ;

渡された親ウィンドウ（*AParent*），コントロール ID（*AnId*），座標（*X, Y*），幅（*W*），高さ（*H*）を使って ***TScrollBar*** のオブジェクトを構築し初期化します。同様の引数を持つ ***TControl*** のコンストラクタを呼び出します。***IsHScrollBar*** が真であれば，***Attr.Style*** で指定されるスタイルに ***SBS_HORZ*** を追加します。偽の場合は，***SBS_VERT*** を追加します。水平スクロールバーの高さまたは垂直スクロールバーの幅として 0 が渡されれば，標準値が使われます。***LineMagnitude*** は 1 に初期化され，***PageMagnitude*** には 10 がセットされます。

関連項目：***TControl*** : : ***TControl***

constructor TScrollBar (PTWindowsObject AParent, int ResourceId, PTModule AModule = NULL) ;

TDialog のスクロールバーコントロールに対応する ***TScrollBar*** のオブジェ

クトを構築します。同じ引数を持つ ***TControl*** のコンストラクタを呼び出します。

引数 ***ResourceId*** は，定義するラジオボタンリソースに必ず対応させます。

constructor TScrollBar (StreamableInit) ; protected

TScrollBar のストリームコンストラクタです。***TControl*** のストリームコンストラクタを呼び出します。入力ストリームのデータを使って ***TScrollBar*** のインスタンスを構築するときに呼び出されます。

関連項目：***TControl*** : : ***TControl***

build static PTStreamable build () ;

TScrollBar (***StreamableInit***)コンストラクタを呼び出します。***TScrollBar*** 型のオブジェクトを構築するとその後でそのデータメンバがストリームから読み込まれます。

DeltaPos int DeltaPos (int Delta) ;

SetPosition を呼び出して，スクロールボックスの位置を *Delta* で渡された値に変更させます。*Delta* が正であればスクロールボックスを下または右に移動します。*Delta* 値が負であれば，スクロールボックスを上または左に移動します。戻り値はスクロールボックスの新しい位置です。

関連項目：***TScrollBar*** : : ***SetPosition***

GetClassName virtual LPSTR GetClassName () ; protected

TScrollBar の Windows 登録クラス名である「SCROLLBAR」を返します。

GetPosition int GetPosition () ;

スクロールボックスの現在の位置を返します。

関連項目：***TScrollBar*** : : ***SetPosition***

GetRange void GetRange (Rint LoVal, Rint HiVal) ;

スクロールボックスの現在の移動許容範囲の両端を ***LoVal*** と ***HiVal*** で返します。

関連項目：***TScrollBar*** : : ***SetPosition, TScrollBar*** : : ***SetRange***

read virtual Pvoid read (Ripstream is) ; protected

TWindow* :: *read を呼び出して基本クラス ***TWindow*** のオブジェクトを読み込ませます。その後で ***LineMagnitude*** と ***PageMagnitude*** を読み込みます。

SBBottom virtual void SBBottom (RTMessage Msg)
= [NF_FIRST + SB_BOTTOM] ; protected

SetPositon を呼び出して，スクロールボックスをスクロールバーの最下端または右端に移動させます。スクロールバーの最下端または最右端までボックスがドラッグされるとこのメンバ関数が呼び出されます。

関連項目：***TScrollBar* :: *SetPositon***

SBLineDown virtual void SBLineDown (RTMessage Msg)
= [NF_FIRST + SB_LINEDOWN] ; protected

SetPosition を呼び出して，スクロールボックスを ***LineMagnitude*** 分下または右に移動させます。スクロールバーの下端または右端にある矢印がクリックされると呼び出されます。

関連項目：***TScroller* :: *SetPosition***

SBLineUp virtual void SBLineUp (RTMessage Msg)
= [NF_FIRST + SB_LINEUP] ; protected

SetPosition を呼び出して，スクロールボックスを ***LineMagnitude*** 分上または左に移動させます。スクロールバーの上端または左端にある矢印がクリックされると呼び出されます。

関連項目：***TScroller* :: *SetPosition***

SBPageDown virtual void SBPageDown (RTMessage Msg)
= [NF_FIRST + SB_PAGEDOWN] ; protected

SetPosition を呼び出して，スクロールボックスを ***PageMagnitude*** 分下または右に移動させます。スクロールバーの下部または右のスクロール領域がクリックされると呼び出されます。

関連項目：***TScroller* :: *SetPosition***

SBPageUp virtual void SBPageUp (RTMessage Msg)
= [NF_FIRST + SB_PAGEUP] ; protected

SetPosition を呼び出して，スクロールボックスを ***PageMagnitude*** 分上または左に移動させます。スクロールバーの上部または左のスクロール領域がクリックされると呼び出されます。

関連項目：***TScroller*** : : ***SetPosition***

SBThumbPosition virtual void SBThumbPosition (RTMessage Msg)
= [NF_FIRST + SB_THUMBPOSITION] ; protected

SetPosition を呼び出してスクロールボックスを移動させます。スクロールボックスが新しい位置に設定されると呼び出されます。

関連項目：***TScroller*** : : ***SetPosition***

SBThumbTrack virtual void SBThumbTrack (RTMessage Msg)
= [NF_FIRST + SB_THUMBTRACK] ; protected

SetPosition を呼び出して，新しい位置へスクロールボックスがドラッグされるのに合わせてスクロールボックスを移動させます。

関連項目：***TScroller*** : : ***SetPosition***

SBTop virtual void SBTop (RTMessage Msg) = [NF_FIRST +
SB_TOP] ; protected

SetPosition を呼び出して，スクロールバーの最上端または右端にスクロールボックスを移動させます。スクロールバーの最上端または最右端までスクロールボックスがドラッグされると呼び出されます。

関連項目：***TScroller*** : : ***SetPosition***

SetPosition void SetPosition (int ThumbPos) ;

スクロールボックスを *ThumbPos* で指定された位置に移動します。*ThumbPos* がスクロールバーの現在の許容範囲外であれば，許容範囲内で最も近い位置に移動します。

関連項目：***TScrollBar*** : : ***GetPosition***

TS
Class

SetRange void SetRange (int LoVal, int Hival) ;

スクロールバーを *LoVal* から *HiVal* までの範囲に設定します。

関連項目：***TScrollBar*** : : ***GetRange***

SetupWindow virtual void SetupWindow () ; protected

スクロールバーの範囲を 0 から 100 までに設定します。この範囲を定義し直すには ***SetRange*** を呼び出します。

関連項目：***TScrollBar*** : : ***SetRange***

Transfer virtual WORD Transfer (Pvoid DataPtr, WORD TransferFlag) ;

DataPtr が指す転送バッファとの間でスクロールバーのデータ（スクロールバー範囲の上限値と下限値，およびスクロールボックスの位置）を転送します。*DataPtr* は第 3 章『**ObjectWindows のその他の構成要素**』に定義されている ***TScrollBarData*** 構造体を指すものとします。

TransferFlag が ***TF_GETDATA*** であれば転送バッファに転送し，***TF_SETDATA*** であれば転送バッファから転送します。

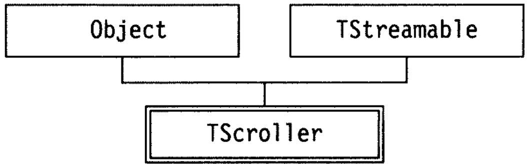
戻り値は，必ず転送データの大きさ，すなわち ***TScrollBarData*** 構造体の大きさです。データを転送せずにデータの大きさを取得するには，*TransferFlag* として ***TF_SIZEDATA*** を渡します。

write virtual void write (Ropstream os) ; protected

TWindow : : ***write*** を呼び出して，基本クラス ***TWindow*** のオブジェクトを書き込ませます。その後で ***LineMagnitude*** と ***PageMagnitude*** を書き込みます。

関連関数

operator＜＜と operator＞＞については 13 ページを参照してください。



TScroller のインスタンスは，通常ウィンドウのスクロールバーを使って ***TWindow*** の表示をスクロールさせます。一方，自動スクロールは，マウスがウィンドウの内から外にドラッグされると，ウィンドウの表示をスクロールします。

TScroller の機能を利用するには，***TWindow*** の下位クラスの ***Scroller*** メンバに，***TWindow*** の下位クラスのコンストラクタでを構築した ***TScroller*** のオブジェクトを設定します。

データメンバ

AutoMode BOOL AutoMode ;

TScroller で「自動スクロール」が使える場合は ***AutoMode*** は真になります。***AutoMode*** が真であれば，マウスがウィンドウの内側から外側にドラッグされるとオーナーウィンドウの表示をスクロールします。***AutoMode*** はデフォルトで真です。

AutoOrg BOOL AutoOrg ;

クライアント領域を描画用に設定するとき，スクローラにクライアント領域の原点を ***XPos*** と ***YPos*** 分だけ自動的にオフセットとするためのフラグです。このフラグはデフォルトで真です。

関連項目：***TScroller*** :: ***BeginView***

ClassHashValue static hashValueType ClassHashValue ; **protected**

TScroller の最後に構築したインスタンスのハッシュ値が入ります。***TScroller*** のインスタンス固有のキーとして使います。

HasHScrollBar BOOL HasHScrollBar ;

オーナーウィンドウに水平スクロールバーがあれば真になります。

HasVScrollBar BOOL HasVScrollBar ;

オーナーウィンドウに垂直スクロールバーがあれば真になります。

InstanceHashValue hashValueType InstanceHashValue ; protected

this のハッシュ値が入ります。***hashValue*** 固有のキーを提供します。

TrackMode BOOL TrackMode ;

スクロールボックスがドラッグされたとき，その移動に合わせてオーナーウィンドウの表示をスクロールする場合は真になります。偽であれば，スクロールボックスのドラッグが終了するまで表示はスクロールされません。デフォルトは真です。

Window PTWindow Window ;

TScroller のオーナーウィンドウのポインタです。

XLine int XLine ;

水平スクロールバーの矢印がクリックされたときに水平方向にスクロールする ***XUnit*** 数です。デフォルト値は 1 です。

XPage int XPage ;

水平スクロールバーのスクロールボックス領域がクリックされたときに水平方向にスクロールする ***XUnit*** 数です。デフォルトではウィンドウの現在の幅（***XUnit***）と同じ値がセットされます。

関連項目：***TScroller*** : : ***SetPageSize***

XPos long XPos ;

スクローラの現在の水平位置を ***XUnit*** 単位で表わしたものです。

XRange long XRange ;

ウィンドウを水平方向スクロールできる総 ***XUnit*** 単位数です。この値はコンストラクタに渡されますが，後で ***SetRange*** を呼び出して後で変更できます。

XUnit int XUnit ;

TScroller が使う水平スクロールの論理スクロール単位です（デバイス単位数で表わします)。この値はコンストラクタに渡されますが，後で *SetUnits* を呼び出して変更できます。

YLine int YLine ;

垂直スクロールバーの矢印がクリックされると垂直方向にスクロールする *YUnit* 数です。デフォルト値は 1 です。

YPage int YPage ;

垂直スクロールバーのスクロールボックス領域がクリックされると垂直方向にスクロールする *YUnit* 単位数です。デフォルトではウィンドウの現在の幅（*YUnit* 単位）と同じ値がセットされます。

関連項目：*TScroller* : : *SetPageSize*

YPos long YPos ;

スクローラの現在の垂直位置を *YUnit* 単位で表わしたものです。

YRange long YRange ;

ウィンドウを垂直方向にスクロールできる総 *YUnit* 単位数です。この値はコンストラクタに渡されますが，後で *SetRange* を呼び出して変更できます。

YUnit int YUnit ;

TScroller が使う垂直スクロールの論理スクロール単位です（デバイス単位数で表わします)。この値はコンストラクタに渡されますが，後で *SetUnits* を呼び出して変更できます。

メンバ関数

constructor TScroller (PTWindow TheWindow, int TheXUnit, int TheYUnit,
long TheXRange, long TheYRange) ;

オーナーウィンドウを *TheWindow*, ***XUnit*** を *TheXUnit*, ***YUnit*** を *TheYUnit*, ***XRange*** を *TheXRange*, ***YRange*** を *TheYRange* とする ***TScroller*** のオブジェクトを構築します。データメンバをデフォルト値で初期化します。***HasHScrollBar*** と ***HasVScrollBar*** はオーナーウィンドウのスクロールバー属性に従って設定します。

constructor TScroller (StreamableInit) ; protected

TScroller のストリームコンストラクタです。入力ストリームのデータで ***TScroller*** のインスタンスを構築するときに呼び出されます。

destructor ~TScroller () ;

TScroller のオブジェクトを破棄します。***Scroller*** に NULL をセットします。

AutoScroll virtual void AutoScroll () ;

ウィンドウの内側から外側にマウスがドラッグされると，その動きに合わせてオーナーウィンドウの表示をスクロールします。表示のスクロール方向と量は，マウスの現在位置に応じて決まります。

BeginView virtual void BeginView (HDC PaintDC, PAINTSTRUCT _FAR & PaintInfo) ;

AutoOrg がデフォルトの真であれば，描画操作中，クライアント領域の論理座標の原点を ***XPos*** と ***YPos*** 分だけ自動的にオフセットします。スクローラが 32767 単位数を超えるなどの理由で ***AutoOrg*** が偽の場合は，手作業でオフセットを設定しなければなりません。

build static PTStreamable build () ;

TScroller (***StreamableInit***)コンストラクタを呼び出します。***TScroller*** 型のオブジェクトを構築すると，そのデータメンバがストリームから読み込まれます。

EndView virtual void EndView () ;

オーナーウィンドウのスクロールバー位置を更新して ***TScroller*** の位置に合わせます。

hashValue virtual hasValueType hasValue () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TScroller*** のインスタンス固有の識別子である ***InstanceHashValue*** を返します。

HScroll virtual void HScroll (WORD ScrollEvent, int ThumbPos) ;

指定された水平 *ScrollEvent* への応答として ***ScrollBy*** または ***ScrollTo*** を呼び出します。スクロールイベントの種類は，対応する Windows の ***SB_*** 定数で特定されます（第 3 章『**ObjectWindows のその他の構成要素**』を参照）。*ThumbPos* には，***SB_THUMBTRACK*** と ***SB_THUMBPOSITION*** の両スクロールイベントがスクローラに通知された時点のスクロールボックスの位置が入ります。

関連項目：***TWindow :: WMHScroll***

isA virtual classType isA () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TScroller*** のクラス識別子である ***scrollerClass*** を返します。

isEqual virtual int isEqual (RCObject testobj) const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。this が指定された *testobj* を指すポインタの場合は真を返します。

IsVisibleRect BOOL IsVisibleRect (long X, long Y, int XExt, int YExt) ;

指定された矩形の一部がオーナーウィンドウ内に現在見えていれば真を返します。

nameOf virtual Pchar nameOf () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TScroller*** のクラス識別文字列「TScroller」を返します。

printOn virtual void printOn (Rostream outputStream) const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。指定されたストリームの ***Window*** の 16 進表現を出力します。

read virtual Pvoid read (Ripstream is) ; protected

AutoMode, AutoOrg, HasHScrollBar, HasVScrollBar, TrackMode, XLine, XPage, XPos, XRange, XUnit, YLine, YPage, YPos, YRange, YUnit を読み込みます。

関連項目：***TScroller*** : : ***write***

ScrollBy void ScrollBy (long Dx, long Dy) ;

ScrollTo を呼び出して，*Dx* と *Dy* 分だけオーナーウィンドウの表示をスクロールさせます。

関連項目：***TScroller*** : : ***ScrollTo***

ScrollTo virtual void ScrollTo (long X, long Y) ;

境界状態をチェックした後で，渡された座標位置までスクロールします。***WMPaint*** メッセージを送らせます。クライアント領域が一部表示されたまま残っていれば，最初にクライアント領域の内容をスクロールします。

関連項目：***TWindow*** : : ***WMPaint***，***TWindow*** : : ***Paint***

SetPageSize virtual void SetPageSize () ;

データメンバ ***XPage*** と ***YPage*** にオーナーウィンドウのクライアント領域の幅と高さを ***XUnit*** 単位と ***YUnit*** 単位でセットします。

関連項目：***TWindow*** : : ***WMSize***

SetRange void SetRange (long TheXRange, long TheYRange) ;

TScroller の ***XRange*** と ***YRange*** に指定された引数をセットします。次に ***SetBarRange*** を呼び出してオーナーウィンドウのスクロールバーをこの値に合わせます。

関連項目：***TScroll*** : : ***SetSBarRange***

SetSBarRange virtual void SetSBarRange () ;

オーナーウィンドウのスクロールバーの範囲を設定して ***TScroller*** の範囲に一致させます。

SetUnits void SetUnits (int TheXUnit, int TheYUnit) ;

データメンバ ***XUnit*** と ***YUnit*** に *TheXUnit* と *TheYUnit* をそれぞれセットします。***SetPageSize*** を呼び出して ***XPage*** と ***YPage*** を更新させます。

VScroll virtual void VScroll (WORD ScrollEvent, int ThumbPos) ;

指定された垂直 *ScrollEvent* への応答として ***ScrollBy*** または ***ScrollTo*** を呼び出します。スクロールイベントの種類は，対応する Windows の ***SB_*** 定数で特定されます（第7章『**Windows 定数**』を参照）。*ThumbPos* には，***SB__THUMBTRACK*** と ***SB_THUMBPOSITION*** の両スクロールイベントがスクローラに通知された時点のスクロールボックス位置が入ります。

関連項目：***TScroller*** :: ***ScrollTo, TWindow*** :: ***WMVScroll***

write virtual void write (Ropstream os) ;

AutoMode, AutoOrg, HasHScrollBar, HasVScrollBar, TrackMode, XLine, XPage, XPos, XRange, XUnit, YLine, YPage, YPos, YRange, YUnit を書き込みます。

XRangeValue long XRangeValue (int AScrollUnit) ;

スクロールバーの水平スクロール値を水平範囲値に変換します。

XScrollValue int XScrollValue (long ARnageUnit) ;

スクロールバーの水平範囲値を水平スクロール値に変換します。

YRangeValue long YRangeValue (int AScrollUnit) ;

スクロールバーの垂直スクロール値を垂直範囲値に変換します。

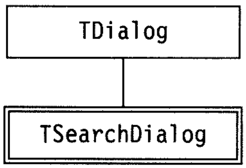
YScrollValue int YScrollValue (long ARangeUnit) ;

スクロールバーの垂直範囲値を垂直スクロール値に変換します。

関連関数

operator＜＜と operator＞＞については 13 ページを参照してください。

TSearchDialog　　　　EDITWND.H



ユーザからの検索/置換用テキストと関連オプションを取得します。*TSearchDialog* のコンストラクタに渡される *TSearchStruct* が，データを両方向に転送します。

STDWNDS.DLG には，検索用ダイアログボックスと置換用ダイアログボックスの 2 つのリソース定義が入っています。*TSearchDialog* を使うには，ダイアログボックスリソースをコンパイルして ObjectWindows アプリケーションに付加します。

関連項目：***TEditWindow*** :: ***CMEditFind,***
TEditWindow :: ***CMEditReplace***

メンバ関数

constructor　TSearchDialog (PTWindowObject AParent, int ResourceId, TSearchStruct _FAR & SearchStruct, PTModule AModule = NULL) ;

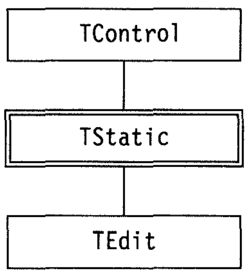
TDialog コンストラクタを呼び出して，指定された親（*AParent*）へのポインタを渡します。*TEdit* と *TCheckBox* の両コントロールオブジェクトを構築して，該当する検索/置換用ダイアログボックスコントロールに対応させます。指定された検索対象の構造体を *TransferBuffer* に設定します。テキストとオ

プションは検索対象の構造体とダイアログボックスの間で転送されます。

引数 *ResourceId* は，STDWNDS.DLG で定義されているダイアログテンプレートに応じて，***SD_SEARCH*** か ***SD_REPLACE*** にしてください。*ResourceId* が ***SD_SEARCH*** であれば検索用 ***TSearchDialog*** が，***SD_REPLACE*** であれば置換用 ***TSearchDialog*** が実行されます。

関連項目：***TWindowsObject***::***TransferBuffer***

TStatic　　STATIC.H



Windows のスタティックテキストインターフェース要素を表わすオブジェクトです。親の ***TWindow*** でスタティックコントロールを作成する場合は必ず使ってください。***TDialog*** のスタティックコントロールに使うとテキストの変更作業が簡単になります。

TS
Class

データメンバ

TextLen　WORD TextLen;

スタティックコントロール用のテキストバッファのサイズを保持します。スタティックコントロールに実際に格納できる文字数は，文字列の末尾を表わすヌルのために，***TextLen*** よりも 1 少ない数になります。***TextLen*** はメンバ関数 ***Transfer*** が転送するバイト数でもあります。

メンバ関数

constructor TStatic (PTWindowsObject AParent, int AnId, LPSTR ATitle, int X, int Y, int W, int H, WORD ATextLen, PTModule AModule = NULL);

指定された親ウィンドウ（*AParent*），コントロールID（*AnId*），テキスト（*ATitle*），親ウィンドウのクライアント領域の原点を基準とする座標位置（*X, Y*），幅（*W*），高さ（*H*），テキスト長（*ATextLen*）を使ってスタティックコントロールオブジェクトを構築します。デフォルトでは，スタティックコントロールは作成と同時に表示され，テキストは左詰めになります。つまり，***Attr.Style*** に ***WS_CHILD*** | ***WS_VISIBLE*** | ***WS_GROUP*** | ***SS_LEFT*** が設定されます。***TControl*** コンストラクタを呼び出します。

関連項目：***TControl*** :: ***TControl***

constructor TStatic (PTWindowsObject AParent, int ResourceId, WORD ATextLen, PTModule AModule = NULL);

TDialog のスタティックコントロールインターフェースコントロールに対応する ***TStatic*** オブジェクトを構築します。同様の引数で ***TControl*** のコンストラクタを呼び出し，***TextLen*** に *ATextLen* を設定します。***DisableTransfer*** を呼び出し，データ転送機構を無効にします。

引数 *ResourceId* は，ユーザが定義するスタティックコントロールリソースに必ず対応させてください。

関連項目：***TControl*** :: ***TControl***

constructor TStatic (StreamableInit); **protected**

TStatic のストリームコンストラクタです。***TControl*** のストリームコンストラクタを呼び出します。入力ストリームからのデータを使って ***TStatic*** のインスタンスを構築するときに呼び出されます。

関連項目：***TControl*** :: ***TControl***

build static PTStreamable build ();

TStatic（***StreamableInit***）コンストラクタを呼び出します。ストリームからデータメンバを読み込む前に ***TStatic*** 型のオブジェクトを構築します。

Clear void Clear () ;

スタティックコントロールのテキストを消去します。

GetClassName virtual LPSTR GetClassName () ; protected

TStatic の Windows 登録クラス名「STATIC」を返します。

GetText int GetText (LPSTR ATextString, int MaxChars) ;

スタティックコントロールのテキストを取得して，それを *MaxChars* サイズの *ATextString* に格納します。コピーした文字数を返します。

GetTextLen int GetTextLen () ;

スタティックコントロールのテキストの長さを返します。

nameOf virtual Pchar nameOf () const ;

コンテナクラスで必要なメンバ関数です。クラス ***Object*** の純粋仮想関数を定義し直します。***TStatic*** のクラス識別文字列「TStatic」を返します。

read virtual Pvoid read (Ripstream is) ; protected

TWindow :: ***read*** を呼び出して基本クラス ***TWindow*** のオブジェクトを読み込ませます。その後で ***TextLen*** を読み込みます。

SetText void SetText (LPSTR ATextString) ;

スタティックコントロールのテキストに *ATextString* で渡された文字列を設定します。

Transfer virtual WORD Transfer (Pvoid DataPtr, WORD TransferFlag) ;

TextLen 文字数分のテキストを，スタティックコントロールと ***DataPtr*** が指す転送バッファとの間で転送します。*TransferFlag* が ***TF_GETDATA*** であれば，テキストはスタティックコントロールからバッファに転送され，***TF_SETDATA*** であれば，スタティックコントロールに転送バッファ内のテキストが設定されます。***Transfer*** はバッファに格納したバイト数，またはバッファから取得したバイト数を示す ***TextLen*** を返します。*TransferFlag* が ***TF_SIZEDATA*** であれば，***TextLen*** を返すだけでデータを転送しません。

TS
Clas

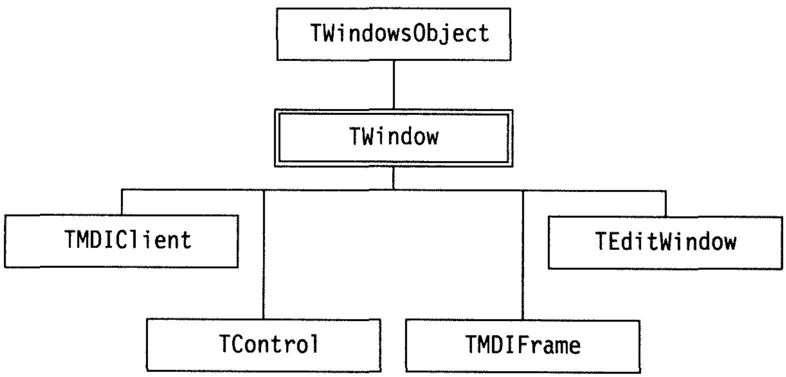
write　virtual void write (Ropstream os) ;

TWindow :: ***write*** を呼び出して，基本クラス ***TWindow*** のオブジェクトを書き出させます。その後で ***TextLen*** を書き出します。

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。

TWindow　　WINDOW.H



TWindowsObject からの派生クラスであり，ウィンドウ固有の動作を提供します。なお，ダイアログボックス固有の動作を提供するのは ***TDialog*** です。***TWindow*** のメンバ関数は，登録や属性など，ウィンドウの作成を指定します。

TWindow はタイトルつきの汎用ウィンドウであり，サイズの変更と移動を行なえます。通常は専用化したウィンドウクラスの基本クラスとして使いますが，インスタンスを構築して作成することもできます。

TMDIFrame を親に指定すれば，***TWindow*** のインスタンスまたは派生クラスを MDI 子ウィンドウとして使うことができます。

TWindow の多くのメンバ関数で，***TWindow*** の表示をスクロールさせる ***TScroller*** オブジェクトを使うためのフックが提供されます。

MDI 対応アプリケーションの専用コントロールである ***TMDIClient*** の基本クラスにもなります。

データメンバ

Attr TWindowAttr Attr ;

TWindowAttr 構造体を保持します。この構造体にはウィンドウの作成属性，すなわちウィンドウオブジェクトに対応するインターフェース要素の特性が入ります。作成属性とはウィンドウのスタイル，拡張スタイル，位置，サイズ，メニューハンドル，コントロール ID のことです。第 3 章『**ObjectWindows のその他の構成要素**』の ***TWindowAttr*** の構造体定義を参照してください。

関連項目：***TWindow*** : : ***TWindow***, ***TWindow*** : : ***Create***

FocusChildHandle HANDLE FocusChildHandle ;

ウィンドウが前回アクティブになったときにフォーカスを持っていた子ウィンドウのハンドルを格納します。

Scroller PTScroller Scroller ;

NULL 以外であれば，***TWindow*** の表示をスクロールさせる ***TScroller*** オブジェクトのポインタです。***TWindow*** のコンストラクタで ***TScroller*** オブジェクトのインスタンスが構築され，***Scroller*** に設定されます。

メンバ関数

constructor TWindow (PTWindowsObject AParent, LPSTR ATitle,
PTModule AModule = NULL) ;

AParent で渡された親ウィンドウでウィンドウオブジェクトを構築します。なお，親を持たないメインウィンドウの場合，*AParent* は NULL になります。対応するテキスト（通常ウィンドウのタイトル）は *ATitle* からコピーします。*AParent* が MDI フレームウィンドウの場合は MDI 子ウィンドウとして作成され，***WB_MDICHILD*** フラグが設定されます。新しい ***TWindow*** がポップアップウィンドウの場合(MDI 子ウィンドウとして作成できない場合)，構築してから ***SetFlags***（***WB_MDICHILD***，FALSE）を呼び出し，その後で作成します。

TWindowsObject のコンストラクタを呼び出します。***Attr*** 構造体の位置フィールドと範囲フィールドに，オーバーラップウィンドウとポップアップウィンドウ用のデフォルトをセットします。*AParent* が NULL の場合は ***Attr.Style*** に ***WS_OVERLAPPEDWINDOW*** をセットします。ただし，***TMDIFrame*** の子である ***TWindow*** の場合は ***WS_CLIPSIBLINGS*** をセットします。その他の ***TWindow*** の場合は ***WS_VISIBLE*** をセットします。

Attr 構造体に設定されたデフォルト値は，***TWindow*** の派生クラスのコンストラクタで設定し直すことができます。また，インターフェース要素の作成前であれば，いつでも設定し直せます。***TWindow*** からの派生クラスのコンストラクタで，***Scroller*** に ***TScroller*** オブジェクトのインスタンスを設定できます(***Scroller*** のデフォルトは NULL です)。その場合，***TWindow*** の派生クラスのコンストラクタでこの ***TWindow*** のコンストラクタを必ず呼び出してください。

関連項目：***TWindowsObject*** : : ***TWindowsObject***

constructor　TWindow (HWND AnHWindow, PTModule AModule = NULL) ;

TWindowsObject のコンストラクタを呼び出して，*AnHWindow* と *AModule* を渡します。

constructor　TWindow (StreamableInit) ;　protected

TWindow のストリームコンストラクタです。***TWindowsObject*** のストリームコンストラクタを呼び出します。入力ストリームのデータで ***TWindow*** のインスタンスを構築するときに呼び出されます。

関連項目：***TWindowsObject*** : : ***TWindowsObject***

destructor　virtual ~TWindow () ;

Attr.Menu を解放し，***Scroller*** が NULL でなければこれを削除します。

ActivationResponse virtual void ActivationResponse (WORD Activated, BOOL IsIconified) ;

WMActivate と ***WMMDIActivate*** によって呼び出され，ウィンドウのコントロールにキーボードインターフェースを提供します。最初に ***TWindowsObject*** : : ***ActivationResponse*** を呼び出します。ウィンドウがアクティブでなくなると（アクティブ状態は 0), 現在フォーカスを持っている子コントロールのハンドルを ***FocusChildHandle*** に保管します。ウィンドウがアクティブになると，ハンドルが格納されている子ウィンドウにフォーカスをセットします。

関連項目：***TWindowsObject*** : : ***EnableKBHandler,***
TWindow : : ***WMMDIActivate***

AssignMenu virtual BOOL AssignMenu (LPSTR MenuName) ;

渡された *MenuName* を ***Attr.Menu*** にセットします。***Attr.Menu*** が前の指していた文字列はすべて解放します。*HWindow* が 0 以外の場合はウィンドウのメニューをロードして設定し，前のメニューがあればこれを廃棄します。

AssignMenu virtual BOOL AssignMenu (int MenuId) ;

メニューリソースファイルに定義されている整数リソース識別子をウィンドウのメニュー ID に割り当てます。その値を，実際に割り当て作業をするもう 1 つの ***AssignMenu*** に渡します。

build static PTStreamable build () ;

TWindow (***StreamableInit***)コンストラクタを呼び出します。***TWindow*** 型のオブジェクトを構築した後にデータメンバをストリームから呼び出します。

関連項目：***TWindow*** : : ***TWindow***

Create virtual BOOL Create () ;

インターフェース要素を作成し，***TWindow*** オブジェクトに対応づけます。ただし，***Status*** が 0 以外の場合，対応づけは行なわれません。まず ***Register*** を呼び出して，ウィンドウクラスが未登録であれば登録させます。次にウィンドウを作成し，対応づけます。なお，***WM_FROMRESOURCE*** フラグが設定されていればインターフェース要素はリソース定義で作成済みなので，対応づけだけを行ない，登録処理と作成処理を飛ばします。次に ***SetupWindow*** を呼び出し，新たに作成したウィンドウを設定します。成功すれば真を返し，失敗すれば ***Error*** を呼び出して偽を返します。

通常 ***Create*** を直接呼び出すことはありません。***Create*** の呼び出しは ***TModule*** : : ***MakeWindow*** がメモリ容量を確認してから行ないます。

関連項目：***TWindowsObject*** : : ***Register***, ***TWindowsObject*** : : ***SetupWindow***, ***TModule*** : : ***MakeWindow***

GetClassName virtual LPSTR GetClassName () ; protected

TWindow のデフォルトの Windows 登録クラス名である「OWLWindow」を返します。

GetWindowClass virtual void GetWindowClass (WNDCLASS _FAR & AWndClass) ; protected

AWndClass で渡された ***WNDCLASS*** 構造体に，***TWindow*** に該当するデフォルトの登録属性をセットします。データメンバ ***Style*** には ***CS_HREDRAW*** か ***CS_VREDRAW*** がセットされます。アイコンは汎用アイコンに，カーソルはストック矢印カーソルに設定されます。バックグラウンド色はシステムのウィンドウバックグラウンド色に設定されます。登録するクラス名は ***GetClassName*** を呼び出して取得します。

関連項目：***TWindow*** : : ***GetClassName***, ***TWindowsObject*** : : ***Register*** , ***TWindow*** : : ***Create***

isA virtual classType isA () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TWindow*** のクラス識別子である ***windowClass*** を返します。

nameOf virtual Pchar nameOf () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TWindow*** のクラス識別文字列「TWindow」を返します。

Paint virtual void Paint (HDC PaintDC, PAINTSTRUCT _FAR & PaintInfo) ; protected

Paint メンバ関数を定義する派生クラスの位置を保持するのに使います。Windows が ***WMPaint*** にウィンドウ内容の再描画を要求すると ***Paint*** が呼び出されます。ディスプレイコンテキストには，テキストとグラフィックス出力関数用の *PaintDC* を使ってください。渡された ***PaintInfo*** 構造体の参照には，描画領域などの描画操作に関する情報が入っています。

関連項目：***TWindow*** : : ***WMPaint***

read virtual Pvoid read (Ripstram is) ;

TWindowsObject : : ***read*** を呼び出して基本クラス ***TWindowsObject*** のオブジェクトを読み込ませます。その後，***WB_FROMRESOURCE*** フラグが立っていなければ，***Attr.Style, Attr.ExStyle, Attr.X, Attr.Y, Attr.W, Attr.H, Attr.Param*** を読み込みます。次に ***Attr.Id, Attr.Menu, Scroller*** を読み込みます。***FocusChildHandle*** に 0 をセットします。

SetupWindow virtual void SetupWindow () ;

新たに作成したウィンドウをセットアップします。***TWindowsObject*** : : ***SetupWindow*** を呼び出して，子リスト内のウィンドウを作成させます。キーボード操作が要求されたとき ***FocusChildHandle*** が設定されていなければ，該当する子ウィンドウをここに設定します。MDI 子ウィンドウの場合は，***SetFocus*** を呼び出して新しいウィンドウにフォーカスを割り当てます。ウィンドウにスクローラオブジェクトがあれば，***SetSBarRange*** を呼び出してスクロールバーの範囲を設定させます。派生クラスで定義し直して，初期化内容を追加することもできます。派生クラスで定義し直す場合は，基本クラスの ***SetupWindow*** を呼び出してください。

関連項目：***TScroller*** : : ***SetBarRange***

WMMDIActivate virtual void WMMDIActivate (RTMessage)
  = [WM_FIRST +WM_MDIACTIVATE] ; protected

DefWndProc を呼び出し，さらに ***WM_MDIACTIVATE*** メッセージを受け取って，メッセージで送られてきたウィンドウハンドルを親（***TMDIFrame***）の ***ActiveChild*** に設定します。***ActivationResponse*** に呼び出して，ウィンドウのコントロールにキーボードインターフェースを提供します。派生クラスで ***WMMDIActivate*** を定義し直す場合は，基本クラスの ***WMMDIActivate*** を必ず呼び出してください。特別な理由がない限り，この呼び出しは他のどの処理よりも後に行ないます。

関連項目：***TWindow*** : : ***ActivationResponse***

WMCreate virtual void WMCreate (RTMessage Msg)
  = [WM_FIRST + WM_CREATE] ; protected

WM_CREATE メッセージを受け取ると ***SetupWindow*** を呼び出します。

関連項目：***TWindow*** : : ***SetupWindow, TWindow*** : : ***Create***

WMHScroll virtual void WMHScroll (RTMessage Msg)
  = [WM_FIRST + WM_HSCROLL] ; protected

ウィンドウの水平スクロールバーイベントに応答します。スクロールバーコントロールからのメッセージの場合は ***DispatchScroll*** を呼び出します。それ以外の場合，***Scroller*** が NULL 以外であれば ***Scroller－＞HScroll*** を呼び出します。(Windows のバグのため，スクロールバーコントロールつきのウィンドウにはスクロールバースタイルがないものとみなされます。)

関連項目：***TScroller*** : : ***HScroll***

WMLButtonDown virtual void WMLButtonDown (RTMessage Msg)
  = [WM_FIRST + WM_LBUTTONDOWN] ; protected

WM_LBUTTONDOWN メッセージに対する応答メンバ関数です。***TWindow*** に自動スクロール用の ***Scroller*** があれば，***WM_LBUTTONUP*** メッセージを受け取るまでループに入り，***Scroller－＞HScroll*** を呼び出します。ループに入る前のマウス入力は取得し，ループに入った後のマウス入力は解放します。マウスクリックを処理するように定義し直しても，定義し直した ***WMLButtonDown*** から呼び出せば，自動スクロール機能はそのまま使えます。

WMMove virtual void WMMove (RTMessage Msg) = [WM_FIRST + WM_MOVE] ; protected

WM_MOVE メッセージを受け取るとウィンドウの新しい位置座標を ***Attr*** 構造体のデータメンバ ***X*** と ***Y*** に保存します。ただし，アイコン化または最大化されていないウィンドウに限ります。

WMPaint virtual void WMPaint (RTMessage Msg) = [WM_FIRST + WM_PAINT] ; protected

Windows の ***WM_PAINT*** メッセージに応答します。ウィンドウに定義済みの Windows クラスがあれば，***DefWndProc*** を呼び出します。それ以外は ***Paint*** を呼び出します。この呼び出しの前と後に Windows が要求する描画用セットアップとクリーンアップ（***BeginPaint*** と ***EndPaint***）を行ないます。***Scroller*** つきのウィンドウでは，***Paint*** の呼び出し前にメンバ関数 ***BeginView*** を呼び出し，呼び出し後にメンバ関数 ***EndView*** を呼び出します。

関連項目：***TMDIClient*** :: ***WMPaint, TWindow*** :: ***Paint, TScroller*** :: ***BeginView, TScroller*** :: ***EndView***

WMSize virtual void WMSize (RTMessage Msg) = [WM_FIRST + WM_SIZE] ; protected

スクローラつきウィンドウの場合，ウィンドウのサイズ変更イベントが発生すると ***SetPageSize*** を呼び出し，新しいウィンドウサイズに合わせます。スクローラつきでもスクローラなしでも，ウィンドウのサイズを ***Attr*** に保管します。ただし，サイズが画面のサイズ以内の場合に限ります。

関連項目：***TScroller*** :: ***SetPageSize***

WMVScroll virtual void WMVScroll (RTMessage Msg) = [WM_FIRST + WM_VSCROLL] ; protected

メッセージの送り元がスクロールバーコントロールであれば，***DispatchScroll*** を呼び出します。それ以外から送られると，***Scroller*** が NULL でなければ，***Scroller*** -> ***VScroll*** を呼び出します。(Windows のバグのため，スクロールバースタイルのあるウィンドウにはスクロールバーコントロールがないものとみなされます。)

関連項目：***TScroller*** :: ***VScroll***

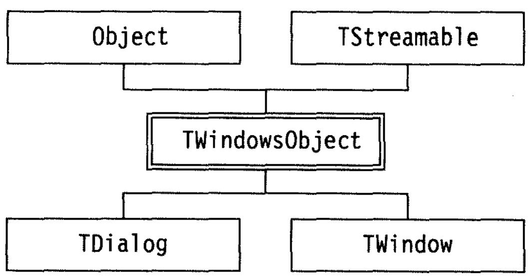
write virtual void write (Ropstream os) ; protected

TWindowsObjet :: ***write*** を呼び出して，基本クラス ***TWindowsObject*** のオブジェクトを書き込ませます。***WB_FROMRESOURCE*** フラグが立っていなければ，***Attr.Style, Attr.ExStyle, Attr.X, Attr.Y, Attr.W, Attr.H, Attr.Param*** を書き込みます。さらに ***Attr.Id, Attr.Menu, Scroller*** を書き込みます。

関連関数

operator＜＜と operator＞＞については，13 ページを参照してください。

TWindowsObject WINDOBJ.H



Object と ***TStreamable*** から派生させた抽象クラスであり，ウィンドウ，ダイアログボックス，コントロールなど，すべての ObjectWindows インターフェースオブジェクトに共通な基本動作を定義します。

また，一連の子ウィンドウを子リストで管理し，子リストに対する動作を提供します。たとえば，***TWindowsObject*** を作成すると，その子ウィンドウも作成されます。***TWindowsObject*** をクローズしようとしても，その子ウィンドウがすべてクローズできなければクローズされません。子リスト内のウィンドウにアクセスするためのメンバ関数も定義しています。

送られてきたメッセージを応答メンバ関数に対応させるときにも ***TWindowsObject*** は重要な役割りを果たします。

データメンバ

DefaultProc FARPROC DefaultProc ; protected

デフォルトのウィンドウ手続きのアドレスを保持します。この手続きはWindows メッセージのデフォルト処理を定義するものです。

HWindow HWND HWindow ;

インターフェースオブジェクトに対応したインターフェース要素へのハンドルを保持します。対応する要素がなければ0になります。対応するインターフェース要素が作成されるとそのハンドルがセットされ，インターフェース要素が破棄されると0になります。

Parent PTWindowsObject Parent ;

このインターフェースオブジェクトの親ウィンドウになるインターフェースオブジェクトのポインタです。

Status int Status ;

インターフェースオブジェクトの初期化で発生したエラーを示します。***Status*** が0以外の値であれば，初期化の失敗を示します。***TWindowsObject*** の派生クラスは，初期化に失敗したオブジェクトにはインターフェース要素を対応づけません。また，インターフェース要素との対応づけに失敗すれば ***Status*** をセットします。

ObjectWindows に定義されているエラー値としては ***EM_INVALIDWINDOW***, ***EM_INVALIDCLIENT***, ***EM_INVALIDCHILD***, ***EM_INVALIDMAINWINDOW*** があります（第3章の「**EM_XXXX 定数**」を参照)。定義したエラー値を ***Status*** にセットして，初期化エラーを示すこともできます。***Status*** のエラー値は ***Create*** または ***SetupWindow*** でセットします。

Title LPSTR Title ;

ウィンドウのタイトルを指すポインタです。

関連項目：***TDialog*** :: ***SetCaption***, ***TDialog*** :: ***SetupWindow***, ***TWindowsObject*** :: ***SetCaption***

TransferBuffer Pvoid TransferBuffer ; protected

TWindowsObject との間でデータを転送するためのバッファを指すポインタです。***TWindowsObject*** は子リスト内のウィンドウが使うデータがバッファに入っているものとみなします。転送データがなければ NULL です。

メンバ関数

constructor TWindowsObject (PTWindowsObject AParent,
PTModule AModule = NULL) ;

Parent に *AParent* をセットします。NULL でなければ，this を *AParent* の子リストに加え，***EnableAutoCreate*** を呼び出して this を *AParent* で作成させ表示させます。

TWindowsObject の下位クラスを DLL に渡すときは，*AModule* で明示的に渡すことをお勧めします。

AModule は ***TWindowsObject*** のインスタンスを所有するアプリケーションまたは DLL モジュールを指定します。*AModule* の値を間違うと ObjectWindows は必要なリソースを探せません。*AModule* がデフォルトの 0 であれば，***TWindowsObject*** は次の規則に従ってモジュールを設定します。なお，このモジュールは ***GetModule*** で取得できます。

- ***TWindowsObject*** のコンストラクタがアプリケーションから呼び出された場合，モジュールはこのアプリケーションに合わせて設定されます。
- ***TWindowsObject*** のコンストラクタが ObjectWindows DLL と動的にリンクされている DLL から呼び出された場合，現在実行中のアプリケーションも同様にリンクされていれば，モジュールは現在実行中のアプリケーションに合わせて設定されます。

  なお，ObjectWindows を DLL として使うためのプログラミング情報は，オンラインの README ファイルにあります。
- ***TWindowsObject*** のコンストラクタが，OWLWL.LIB などの ObjectWindows ライブラリに静的にリンクされている DLL から呼び出される場合，または，呼び出し側の DLL が ObjectWindows の DLL に動的にリンクされてはいるが現在実行中のアプリケーションがそうなっていない場合は，モジュールの設定にデフォルトは使えません。***Status*** に ***EM_INVALIDMODULE***

がセットされ，オブジェクトの作成は失敗となります。

関連項目：***TWindowsObject*** :: ***EnableAutoCreate***

constructor TWindowsObject (STreamableInit) ; protected

TWindowsObject のストリームコンストラクタです。入力ストリームのデータを使って派生クラスのインスタンスを構築するときに呼び出されます。

destructor virtual ~TWindowsObject () ;

Destroy を呼び出して，対応づけられたまま残っているインターフェース要素を破棄します。子リスト内のウィンドウオブジェクトを削除し，さらに ***Parent*** の子リストから this を除去します。

ActivationResponse virtual void ActivationResource (WORD Activated, BOOL IsIconified) ;

TWindowsObject :: ***WMActivate*** と ***TWindow*** :: ***WMMDIActivate*** から呼び出されます。this がアクティブ状態のゼロ以外であり，同時に this のキーボード処理が要求されていれば，***GetApplication***() - > ***SetKBHandler*** を呼び出してキーボード処理を有効にします。それ以外はキーボード処理を無効にします。ウィンドウが最小化されていれば ***IsIconified*** は真になり，最小化されていなければ偽になります。

関連項目：***TWindowsObject*** :: ***WMActivate***, ***TWindowsObject*** :: ***EnableKBHandler***, ***TApplication*** :: ***SetKBHandler***, ***TWindow*** :: ***ActivationResponse***

build static PTStreamable build () ;

TWindowsObject(***StreamableInit***)のコンストラクタを呼び出します。***TWindowsObject*** 型のオブジェクトを構築すると，そのデータメンバがストリームから読み込まれます。

AfterDispatchHandler virtual void AfterDispatchHandler () ;

送られてきたメッセージの後処理を行なうように定義し直してください。デフォルトのままでは何もしません。メッセージ応答の起動後に ***DispatchAMessage*** から呼び出されます。

関連項目：***TWindowsObject*** :: ***DispatchAMessage***

BeforeDispatchHandler virtual void BeforDispatchHandle () ;

送られてきたメッセージの前処理を行なうように定義し直してください。デフォルトのままでは何もしません。メッセージの起動前に ***DispatchAMessage*** から呼び出されます。

関連項目：***TWindowsObject*** : : ***DispatchAMessage***

CanClose virtual BOOL CanClose () ;

対応したインターフェース要素をクローズできる場合は真を返します。各子ウィンドウのメンバ関数 ***CanClose*** を呼び出します。この ***CanClose*** の呼び出しに対し子ウィンドウから1つでも偽が返されれば，偽を返します。

ChildWithid PTWindowsObject ChildWithId (int Id) ;

子ウィンドウリスト内のウィンドウのうち，指定された ID を持つウィンドウのポインタを返します。指定 ***Id*** を持つ子ウィンドウがなければ，NULL を返します。

CloseWindow void CloseWindow () ;

ShutDownWindow を呼び出しても問題がないことを確かめてから ***ShutDownWindow*** を呼び出します。this がアプリケーションのメインウィンドウであれば，***GetApplication***() - > ***CanClose*** を呼び出します。メインウィンドウでなければ，this - > ***CanClose*** を呼び出してウィンドウをクローズできるかどうかを判定します。

関連項目：***TWindowObject*** : : ***CanClose, TApplication*** : : ***CanClose, ShutDownWindow, WMClose***

CMExit virtual void CMExit (RTMessage Msg)
　= [CM_FIRST + CM_EXIT] ; protected

CM_EXIT という ID を持つメニュー項目が選択されると呼び出されます。**this** がメインウィンドウであれば，***CloseWindow*** を呼び出します。

Create virtual BOOL Create () = 0 ;

Windows のインターフェース要素に対応する ObjectWindows のインターフェースオブジェクトを作成するため，派生クラスで定義し直す純粋仮想関数です。

CreateChildren BOOL CreateChildren () ;

子リスト内の子ウィンドウのうち，***WB_AUTOCREATE*** マスクによって自動作成フラグがセットされている子ウィンドウを作成します。

関連項目：***TWindowsObject*** : : ***EnableAutoCreate,*** ***TWindowsObject*** : : ***DisableAutoCreate***

DefChildProc virtual void DefChildProc (RTMessage Msg) ;

送られてきた子 ID メッセージのデフォルト処理を行ないます。デフォルトで ***DefWndProc*** を呼び出します。

このメンバ関数を定義し直すと，子 ID 応答が定義されていない子ウィンドウからのメッセージに応答させることができます。このほうが，子 ID 応答メンバ関数を各子ウィンドウに定義するより便利な場合があります。

DefCommandProc virtual void DefCommandProc (RTMessage Msg) ; protected

送られてきたコマンドメッセージのデフォルト処理を行ないます。メッセージの本来の受け手を探し，それがこのウィンドウであれば，***DefWndProc*** を呼び出します。本来の受け手が親ウィンドウであれば，メッセージを親に渡します。それ以外は本来の受け手に渡します。

このメンバ関数を定義し直すと，コマンド応答メンバが定義されていないコマンドメッセージに応答させることができます。このほうが，コマンド応答メンバ関数を各コマンドに定義するよりも便利な場合があります。

DefNotificationProc virtual void DefNotificationProc (RTMessage Msg) ; protected

this が生成した通知メッセージのデフォルト処理を行ないます。メッセージは，子 ID メッセージとして親に渡します。なお，this は自分自身からの通知メッセージを受け取って処理を行なうこともできます。

このメンバ関数を定義し直すと，this から送られたすべての通知メッセージに応答させることができます。この応答方法のほうが，各通知メッセージに通知応答メンバ関数を定義するよりも便利な場合があります。

DefWndProc virtual void DefWndProc (RTMessage Msg) ;

送られてきたメッセージのデフォルト処理を指定します。Windows で定義されているデフォルト処理を起動します。ウィンドウのデフォルトウィンドウ手続きを呼び出して，*Msg.Result* に戻り値をセットします。this がダイアログボックスであれば，*Msg.Result* に 0 をセットして，Windows にデフォルト処理を起動させます。

Destroy virtual void Destroy () ;

子リスト内の各ウィンドウに対して ***EnableAutoCreate*** を呼び出した後で対応するインターフェース要素を破棄します。これで，this が作成し直されると，子リストのウィンドウも確実に作成し直されます。

関連項目：***TWindowsObject*** : : ***WMDestroy,*** ***TWindowsObject*** : : ***EnableAutoCreate***

DisableAutoCreate void DisableAutoCreate () ;

子ウィンドウインターフェース要素をその親ウィンドウと同時に作成し表示する機能を無効にします。ポップアップウィンドウとコントロールの作成および表示を親ウィンドウより後にする場合に呼び出してください。

関連項目：***TWindowsObject*** : : ***EnableAutoCreate***

DisableTransfer void DisableTransfer () ;

インターフェースオブジェクトの転送機構を無効にして，転送バッファとの間でステータス情報を転送できないようにします。

DispatchAMessage virtual void DispatchAMessage (WORD AMsg, RTMessage AMessage,
void (TWindowObject : :* _FAR)
(RTMessage)) ;

送られてきた Windows メッセージを正しい応答メンバ関数に割り当てます。定義されている応答がなければ，かわりにデフォルトのメンバ関数を呼び出します。デバッグ中にブレークポイントを設定するときに便利です。

メッセージを割り当てる前に ***BeforeDispatchHandler*** を呼び出し，割り当てた後で ***AfterDispacthHandler*** を呼び出します。

DispatchScroll void DiapatchScroll (RTMessage Msg) ; protected

WMHScroll と ***WMVScroll*** から呼び出され，スクロールバーコントロールからのメッセージを届けます。

関連項目：***TWindowsObject*** :: ***WMHScroll,***
TWindowsObject :: ***WMVScroll***

DrawItem virtual void DrawItem (DRAWITEMSTRUCT_FAR & DrawInfo) ;

定義し直すと，描画可能な子コントロールまたはメニュー項目の描画方法を指定できます。メニュー項目や自分の描画方法を知らないコントロールを描画するときに呼び出されます。コントロールや項目が選択されたとき，またはフォーカスが移動したときも呼び出され，新しい状態に応じた外観に変更します。該当する関数を定義し直せば，描画可能なコントロールに自分自身を描画させることもできます。

EnableAutoCreate void EnableAutoCreate () ;

対応する子ウィンドウインターフェース要素を親ウィンドウと同時に作成し表示できるようにします。この機能はデフォルトではウィンドウとコントロールに有効であり，ダイアログには無効です。ダイアログボックスを親ウィンドウと同時に作成し表示するには ***EnableAutoCreate*** を呼び出します。

関連項目：***TWindowsObject*** :: ***DisableAutoCreate***

EnableKBHandler void EnableKBHandler () ;

ウィンドウまたはモードなしダイアログボックスのコントロールのキーボードインターフェースを有効にし，ユーザがタブや矢印キーでコントロール間を移動できるようにします。デフォルトでは，キーボードインターフェースはウィンドウとダイアログボックスには無効です。Windows ではこの機能はダイアログボックスに自動的に提供されます。

関連項目：***TWindowsObject*** :: ***WMActivate,***
TAppliation :: ***SetKBHandler***

EnableTransfer void EnableTransfer () ;

転送機構を有効にして，ウィンドウと転送バッファとの間で状態情報を転送できるようにします。

FirstThat PTWindowsObject FirstThat (TCondFunc Test, Pvoid, PParmList) ;

子ウィンドウリストに対し繰り返し処理を実行し，*Test* 論理関数を呼び出して順番に各子ウィンドウを引数として *PParamList* とともに渡します。*Test* を呼び出して真が返れば，繰り返し処理は停止し，***FirstThat*** は *Test* に渡された子ウィンドウオブジェクトを返します。偽が返ると ***FirstThat*** は NULL を返します。***TCondFunc*** については，第 3 章**『ObjectWindows のその他の構成要素』**を参照してください。

次の例では，***GetFirstChecked*** が ***FirstThat*** を呼び出して，子リストの最初にチェックされたチェックボックスを指すポインタ(*P*)を取得させます。

```
BOOL IsThisChecked (Pvoid P, Pvoid)  {
    return ((PTCheckBox)P - >GetCheck () == BF_CHECKED) ;
}
PTCheckBox TMyWindow : : GetFirstChecked() ;  {
    return (FirstThat (IsThisBoxChecked, NULL))
}
```

FirstThat PTWindowObject FirstThat (TCondMemFunc Test, Pvoid PParamList) ;

上の ***FirstThat*** を参考にしてください。異なる点は，上の ***FirsThat*** が引数として関数を使うのに対して，こちらはメンバ関数を使うことです。***TCondMemFunc*** については，第 3 章**『ObjectWindows のその他の構成要素』**を参照してください。

ForEach void ForEach (TActionFunc Action, Pvoid PParamList) ;

子ウィンドウリストに対し繰り返し処理を行ない，*Action* として渡された関数を呼び出し，子ウィンドウを順に引数として *PParamList* とともに渡します。***TActionFunc*** については，第 3 章**『ObjectWindows のその他の構成要素』**を参照してください。

次の例では，***CheckAllBoxes*** が ***ForEach*** を呼び出して，子リスト内のすべてのチェックボックスをチェックしています。

```
void CheckTheBox (Pvoid p, Pvoid)  {
    (PTCheckBox)P - >Check () ;
}
void CheckAllBoxes() ;  {
```

```
        ForEach (CheckTheBox, NULL) ;
    }
```

ForEach void ForEach (TActionMemFunc Action, Pvoid PParamList) ;

上の ***ForEach*** を参考にしてください。異なる点は，上の ***ForEach*** が関数を使うのに対して，こちらはメンバ関数を使うことです。***TActionMemFunc*** については，第 3 章『**ObjectWindows のその他の構成要素**』を参照してください。

GetApplication PTApplication GetApplication () ;

this に対応する ***TApplication*** オブジェクトのポインタを取得します。

GetChildPtr void GetChildPtr (Ripstream is, RPTWindowsObject P) ; protected

子ウィンドウを指すポインタの参照を，指定されたストリームから読み出します。***PutChildPtr*** が書き込んだ子ウィンドウのポインタを読み込むときに呼び出してください。

関連項目：***TWindowsObject*** : : ***GetSiblingPtr,***
TWindowsObject : : ***PutSiblingPrt, TWindowsObject*** : : ***PutChildPtr***

GetChildren void GetChildren (Ripstream is) ;

指定されたストリームから子ウィンドウを読み込んで，子ウィンドウリストに入れます。***PutChildren*** がストリームに書き込んだ子ウィンドウを読み込む ***read*** 操作中にだけ呼び出すことができます。

☞ このメンバ関数は，子リストを現在空とみなします！

GetClassName virtual LPSTR GetClassName () = 0 ; protected

派生クラスで必ず定義し直さなければならない純粋仮想関数です。Windows の登録クラス名を返します。

関連項目：***TWindowsObject*** : : ***GetWindowClass***

GetClient virtual PTMDIClient GetClient () ;

MDI クライアントウィンドウを持たないすべての非 MDI インターフェースオブジェクトに対して NULL を返します。MDI クライアントウィンドウのポインタを返す場合は，***TMDIWindow*** で定義し直します。

GetFirstChild PTWindowsObject GetFirstChild () ;

インターフェースオブジェクトの子リスト内の最初の子ウィンドウを指すポインタを返します。

GetId virtual int GetId () ;

指定された親の子リストでウィンドウを検索するために使った ***Id*** を返します。***TControl*** の下位クラスで定義し直すと，オブジェクトのリソース識別子である ***Attr.Id*** を返します。

デフォルトとして 0 を返します。この ID が 0 の非コントロールウィンドウは表示されません。個々のウィンドウをアドレス指定する場合は 0 以外を返すように定義し直します。

関連項目：***TControl*** :: ***GetId***

GetInstance FARPROC GetInstance () ;

ウィンドウのインスタンスサンクを返します。

GetLatChild PTWindowsObject GetLastChild () ;

インターフェースオブジェクトの子リストの最後の子ウィンドウを指すポインタを返します。

GetModule PTModule GetModule () ;

このオブジェクトを所有する ***TModule*** を指すポインタを返します。

GetSiblingPtr void GetSiblingPTr (Ripstream is, RPTWindowsObject P) ; protected

指定されたストリームから兄弟ウィンドウのポインタの参照を読み出します。**兄弟ウィンドウ**とは親が同じウィンドウのことです。たとえば，***TCheckBox*** の ***TGroupBox*** は ダイアログボックスの ***TCheckBox*** の兄弟です。***GetSiblingPtr*** は ***PutSiblingPtr*** を呼び出して書き込んだ子ポインタ値を読み出す ***read*** 操作以外では使わないでください。***P*** にロードされた値は，ウィンドウの親が ***read*** 操作を完了するまでは有効になりません。したがって，ストリームコンストラクタ内で兄弟ウィンドウのポインタを逆参照しても正しい結果は得られません。

関連項目：***TWindowsObject*** :: ***PutSiblingPtr,***

TWindowsObject* : : *GetChildPtr, TWindowsObject* : : *PutChildPtr

GetWindowClass virtual void GetWindowClass (WNDCLASS _FAR & AWndClass) ; protected

派生クラスで定義し直すと，指定された Windows 登録クラス構造体の登録属性を変更できます。

hashValue virtual hasValueType hasValue () const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。***TWindowsObject*** のハッシュ値である ***HWindow*** を返します。

isA virtual classType isA () const = 0 ;

純粋仮想関数です。派生クラスで定義し直すと固有のクラス識別子を返せます。

isEqual virtual int isEqual (RCObject testwin) const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。this が指定された *testwin* を指すポインタであれば，真を返します。

IsFlagSet BOOL IsFlagSet (WORD Mask) ;

指定された *Mask* のビットフラグの状態を返します。ビットフラグがセットされていれば真を返し，セットされていなければ偽を返します。

関連項目：***TWindowsObject* : : *SetFlags***

nameOf virtual Pchar nameOf () const = 0 ;

純粋仮想関数です。派生クラスで定義し直すと，クラス識別文字列を返せます。

Next PTWindowObject Next () ;

親ウィンドウの子ウィンドウリスト内の次のウィンドウのポインタを返します。当該ウィンドウがリストに最後に追加されたウィンドウであれば，最初に追加された子ウィンドウのポインタを返します。

関連項目：***TWindowsObject* : : *Previous***

printOn virtual void printOn (Rostream outputStream) const ;

コンテナクラスで必要とします。***Object*** クラスの純粋仮想関数を定義し直します。指定されたストリームに ***HWindow*** の 16 進表現を出力します。

Previous PTWindowsObject Previous () ;

親ウィンドウの子ウィンドウリスト内の前のウィンドウのポインタを返します。

関連項目：***TWindowsObject*** : : ***Next***

PutChildPtr void PutChildPtr (Ropstream os, PTWindowsObject P) ; protected

指定されたストリームに子ウィンドウを書き込みます。***PutChildPtr*** は子ウィンドウポインタを書き込む ***write*** 操作以外では呼び出さないでください。書き込まれた子ウィンドウポインタは後で ***GetChildPtr*** で読み出せます。

関連項目：***TWindowsObject*** : : ***GetSiblingPtr,***
TWindowsObject : : ***PutSiblingPtr, TWindowsObject*** : : ***GetChildPtr***

PutChildren void PutChildren (Ropstream os) ;

子ウィンドウリストの子ウィンドウを指定されたストリームに書き込みます。***PutChildren*** は子ウィンドウを書き込む ***write*** 操作以外では呼び出さないでください。書き込まれた子ウィンドウは後で ***GetChildren*** で読み出すことができます。

PutSiblingPtr void PutSiblingPtr (Ropstream os, PTWindowsObject P) ; protected

兄弟ウィンドウのポインタを指定されたストリームに書き込みます。兄弟ウィンドウとは親が同じウィンドウのことです。***PutSiblingPtr*** は兄弟ウィンドウのポインタを書き込む ***write*** 操作以外では使わないでください。書き込まれた兄弟ウィンドウのポインタは後で ***GetSiblingPtr*** で読み出すことができます。

関連項目：***TWindowsObject*** : : ***GetSiblingPtr,***
TWindowsObject : : ***GetChildPtr, TWindowsObject*** : : ***PutChildPtr***

read virtual Pvoid read (Ripstream is) ; protected

子ウィンドウリスト，***HWindow, Parent, TransferBuffer*** に 0 をセットします。オブジェクトのインスタンスを作成します。***Title***，***Status***，フラグ，***CreateOrder*** を読み込みます。子ウィンドウを読み込むには ***GetChildren*** を呼び出します。

Register virtual BOOL Register () ;

this の Windows 登録クラスが未登録であれば，登録します。***GetClassName*** と ***GetWindowClass*** を呼び出して，this の Windows 登録クラス名と属性を取得します。this が登録されていれば真を返します。

関連項目：***TWindowsObject*** : : ***GetClassName,*** ***TWindowsObject*** : : ***GetWindowClass***

RemoveClient void RemoveClient () ; protected

指定されたクライアントウィンドウを子ウィンドウリストから除去します。クライアントウィンドウは兄弟ウィンドウとは別に処理してください。

SetCaption void SetCaption (LPSTR ATitle) ;

Title が指す割り当てられた文字列に *ATitle* をコピーします。インターフェース要素のタイトルに *ATitle* をセットします。前のタイトルがあれば削除します。

SetFlags void SetFlags (WORD Mask, BOOL OnOff) ;

OnOff で渡された論理引数に従って，ビットフラグをオンまたはオフにします。真が渡されれば，*Mask* のビットがセットされます。偽が渡されれば，ビットはクリアされます。*Mask* は ***WB***_定数のいずれか，または組み合わせとします。

関連項目：***TWindowsObject*** : : ***IsFlagSet***

SetParent virtual void SetParent (PTWindowsObject NewParent) ;

指定された親ウィンドウオブジェクトを ***Parent*** にセットします。前の親ウィンドウの子ウィンドウリストから this があれば除去し，this を新しい親の子ウィンドウリストに加えます。

SetTransferBuffer void SetTransferBuffer (Pvoid ATransferBuffer) ;

TransferBuffer に *ATransferBuffer* をセットします。

SetupWindow virtual void SetupWindow () ; protected

対応するインターフェース要素の作成に続いてセットアップをします。子ウィンドウリストに対して繰り返し処理を行ない，自動作成機能が有効な各子ウィンドウオブジェクトに対応するインターフェース要素を作成します。この自動作成機能は，デフォルトではウィンドウとコントロールに有効ですが，ダイアログボックスには無効です。子ウィンドウを作成できない場合は ***Status*** に ***EM_INVALIDCHILD*** がセットされます。この後で ***SetupWindow*** は ***TransferData*** を呼び出します。派生クラスで定義し直すと，さらに専用的な初期化を行なえます。

Show void Show (int ShowCmd) ;

ShowCmd で渡された値が指定する方法で，インターフェース要素を画面に表示します。以下の値は *ShowCmd* が取り得る値です。

値	記述
SW_HIDE	非表示
SW_SHOW	ウィンドウの現在のサイズと位置で表示
SW_MAXIMIZE	最大化してアクティブ状態
SW_MINIMIZED	最小化してアクティブ状態
SW_SHOWNORMAL	復元してアクティブ状態

以上の値は WINDOWS.H で定義されます。

ShutDownWindow virtual void ShutDownWindow () ;

対応するインターフェース要素を無条件に破棄して，this を削除します。

Transfer virtual WORD Transfer (Pvoid DataPtr, WORD TransferFlag) ;

指定された転送バッファとの間でウィンドウデータを転送します。*TransferFlag* は，指定されたバッファでデータを読み出すのか，データを書き込むのか，または転送データのサイズだけを返すのかを指定します。戻り値は，転送データのサイズ（バイト数）です。このメンバ関数は 0 を返すだけであり，***TControl*** の派生クラスで定義し直せます。

TransferData virtual void TransferData (WORD Direction) ;

TransferBuffer が NULL でなければ，インターフェースオブジェクトの転送対象の子ウィンドウとバッファとの間でデータを転送します。***WB_TRANSFER*** フラグがセットされていない子ウィンドウとの間ではデータは転送されません。転送に関与する各子ウィンドウのメンバ関数 ***Transfer*** を呼び出して，転送データを指すポインタと *Direction* が指定する方向（***TF_SETDATA, TF_GETDATA, TF_SIDEDATA*** のいずれか）を渡します。

関連項目：***TWindowsObject*** : : ***EnableTransfer, TWindowsObject*** : : ***DisableTransfer, TWindowsObject*** : : ***SetupWindow***

WMActivate virtual void WMAction (RTMessage Msg)
= [WM_FIRST + WM_ACTIVATE] ; protected

WM_ACTIVATE メッセージが送られてくると，***ActivationResponse*** を呼び出して，キーボード処理を有効または無効にします。

関連項目：***TApplication*** : : ***SetKBHandler, TWindowsObject*** : : ***EnableKBHandler***

WMClose virtual void WMClose (RTMessage Msg)
= [WM_FIRST + WM_CLOSE] ; protected

ウィンドウのクローズ要求があると ***CloseWindow*** を呼び出します。

関連項目：***TWindowsObject*** : : ***Destroy***

WMCommand virtual void WMCommand (RTMessage Msg)
= [WM_FIRST + WM_COMMAND] ; protected

コマンドメッセージ，子 ID メッセージ，通知メッセージを届け，該当する応答メンバ関数を呼び出します。

WMDestroy virtual void WMDestroy (RTMessage Msg)
= [WM_FIRST + WM_DESTROY] ; protected

WM_DESTROY メッセージが送られてくると，this がメインウィンドウであれば，終了メッセージを送ってアプリケーションを終了させます。

WMDrawItem virtual void WMDrawItem (RTMessage Msg)
= [WM_FIRST + WM_DRAWITEM] ; protected

コントロールのメンバ関数 ***WMDrawItem*** を呼び出して，描画可能なコントロールに対する ***WM_DRAWITEM*** メッセージを送信します。このメッセージが描画可能なメニュー項目に対するものであれば，***WMDrawItem*** を呼び出します。

関連項目：***TWindowsObject*** :: ***DrawItem, TControl*** :: ***WMDrawItem***。複数の形式があるストリームクラスのメンバ関数についてはオンライン Help を参照してください。

WMHScroll virtual void WMHScroll (RTMessage Msg)
= [WM_FIRST + WM_HSCROLL] ; protected

WM_HSCROLL メッセージが送られてくると ***DispatchScroll*** を呼び出します。

関連項目：***TWindowsObject*** :: ***DispatchScroll***

WMNCDestroy virtual void WMNCDestroy (RTMessage Msg)
= [WM_FIRST + WM_NCDESTROY] ; protected

Windows インターフェース要素に送られる最後のメッセージである ***WM_NCDESTROY*** に応答します。*HWindow* を 0 にセットして，インターフェース要素が this とはもはや対応していないことを示します。this が別名のウィンドウオブジェクトであれば，this を削除します。

WMQueryEndSession virtual void WMQueryEndSession (RTMessage Msg)
= [WM_FIRST + WM_QUERYENDSESSION] ; protected

Windows がクローズしようとすると応答します。this がメインウィンドウであれば，***GetApplication***() - > ***CanClose*** の呼び出しの結果を返します。this がメインウィンドウでなければ，this - > ***CanClose*** の呼び出しの結果を返します。

WMVScroll virtual void WMVScroll (RTMessage Msg)
= [WM_FIRST + WM_VSCROLL] ; protected

WM_VSCROLL メッセージが送られてくると ***DispatchScroll*** を呼び出します。

関連項目：***TWindowsObject : : DispatchScroll***

write virtual void write (Ropstream os) ; protected

Title, Status, フラグ，***CreateOrder*** を書き込みます。***PutChildren*** を呼び出して子ウィンドウを書き込みます。

TW
Class

第2章

ストリームクラス

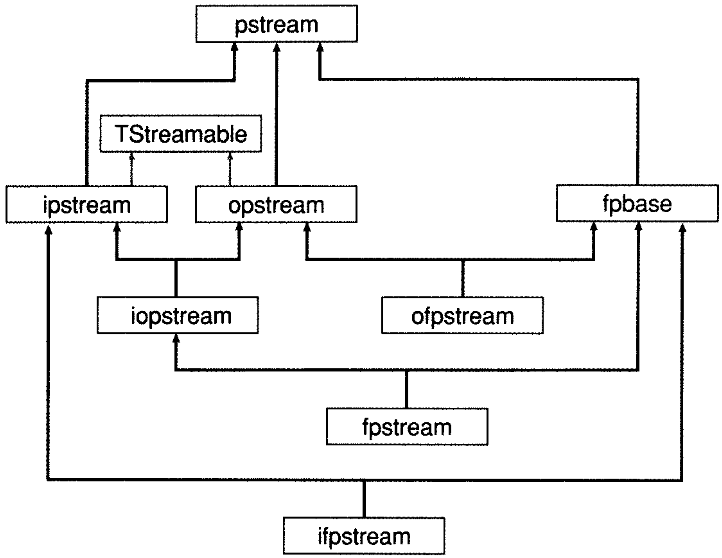
この章は，ストリーム型オブジェクトをサポートするすべてのパブリッククラスをアルファベット順に記載し，その用途，メンバ，演算子，フレンドを解説しています。

特定のクラスに関する情報を探す方法は，第1章の7ページで説明しています。また，9ページにあるサンプルクラスの記載形式も参考にしてください。

ストリームの階層

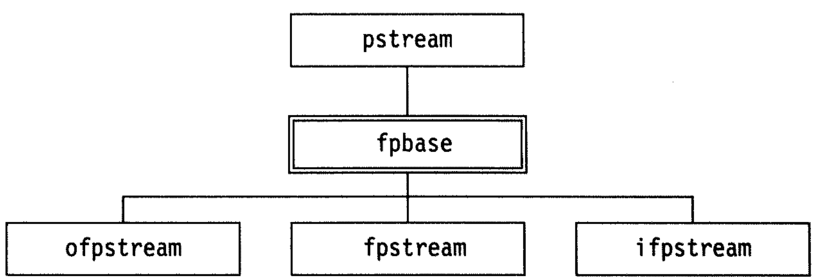
ObjectWindows で使うライブラリは，ストリーム型オブジェクトに関連するクラス群が階層構造になったものです。この章は，**図 2.1** に示す各クラスの説明です。

図 2.1
ObjectWindows で使うストリームクラスの階層構造



➡ ***TStreamable*** に向かう灰色の矢印は，このクラスがフレンドクラスであることを示しています。

fpbase　　OBJSTRM.H



fpbase はすべてのオブジェクトファイルのストリーム I/O に共通な基本操作を提供します。

メンバ関数

constructor fpbase () ;
fpbase (PCchar name, int omode, int prot = filebuf : : openprot) ;
fpbase (int f) ;
fpbase (int f, Pchar b, int len) ;

バッファリングつきの ***fpbase*** オブジェクトを作成します。バッファのサイズと初期内容は，引数 *len* と *b* で設定します。ファイル名，モード，保護(*prot*)の各引数を指定するか，またはファイル記述子 *f* を使うと，ファイルをオープンしてストリームに結びつけることができます。

destructor ~fpbase () ;

fpbase オブジェクトを破棄します。

attach void attach (int f) ;

可能であれば記述子 *f* のファイルをこのストリームに結びつけます。結果に従って，**ios : : state** を設定します。

close void close () ;

ストリームおよび結びついているファイルをクローズします。

open void open (PCchar name, int mode, int prot = filebuf : : openprot) ;

指定されたファイルを，指定された *mode*（*app, ate, in, out, binary, trunc, nocreate, noreplace* のいずれか）と保護でオープンします。オープンしたファイルはこのストリームに結びつけられます。

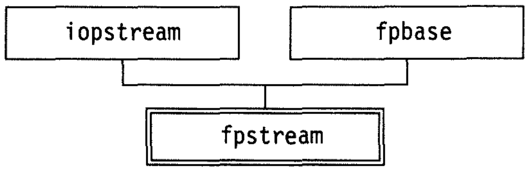
rdbuf Pfilebuf rdbuf () ;

現在のファイルバッファのポインタを返します。

setbuf void setbuf (Pchar buf, int len) ;

サイズ *len* のバッファを確保します。

fpstream　　OBJSTRM.H



fpstream は，基本クラス ***fpbase*** と ***iopstream*** を単にまとめたものです。双方向ファイルストリームにストリーム型オブジェクトを同時に読み書きする基本クラスを規定します。

メンバ関数

constructor　fpstream () ;

fpstream (PCchar name, int mode, int prot = filebuf : : openprot) ;

fpstream (int f) ;

fpstream (int f, Pchar b, int len) ;

バッファリングつきの ***fpstream*** オブジェクトを作成します。バッファのサイズと初期内容は，引数 *len* と *b* で設定します。ファイル名，モード，保護の各引数を指定するか，またはファイル記述子 *f* を使うと，ファイルをオープンしてストリームに結びつけることができます。

destructor　~fpstream () ;

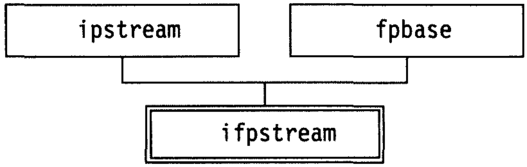
fpstream オブジェクトを破棄します。

open　void open (PCchar name, int mode, int = prot filebuf : : openprot) ;

指定されたファイルを，指定された *mode*（*app, ate, in, out, binary, trunc, nocreate, noreplace* のいずれか）と保護でオープンします。オープンしたファイルはこのストリームに結びつけられます。

rdbuf　Pfilebuf rdbuf () ;

データメンバ ***bp*** を返します。



ifpstream は，基本クラス ***fpbase*** と ***ipstream*** を単にまとめたものものです。ファイルストリームからストリーム型オブジェクトを読み出す（抽出する）基本クラスを提供します。

メンバ関数

constructor

ifpstream () ;
ifpstream (PCchar name, int mode = ios : : in,
int prot = filebuf : : openprot) ;
ifpstream (int f) ;
ifpstream (int f, Pchar b, int len) ;

バッファリングつきの ***ifpstream*** オブジェクトを作成します。バッファのサイズと初期内容は，引数 *len* と *b* で設定します。ファイル名，モード，保護の各引数を指定するか，またはファイル記述子 *f* を使うと，ファイルをオープンしてストリームに結びつけることができます。

destructor

~ifpstream () ;

ifpstream オブジェクトを破棄します。

open

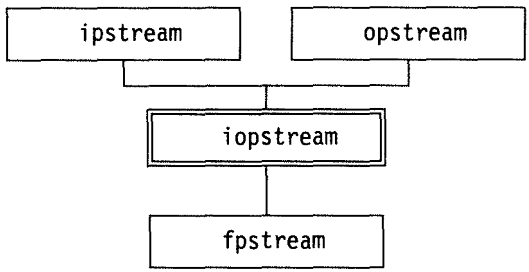
void open (PCchar name, int mode = ios : : in, int prot = filebuf : : openprot) ;

指定されたファイルを，指定された *mode*（*app, ate, in, out, binary, trunc, nocreate, noreplace* のいずれか）と保護でオープンします。デフォルトは，*in*（入力）と *openprot* プロテクトです。オープンしたファイルはこのストリームに結びつけられます。

rdbuf　Pfilebuf rdbuf () ;

現在のファイルバッファのポインタを返します。

iopstream　OBJSTRM.H



iopstream は，基本クラス ***opstream*** と ***ipstream*** を単にまとめたものです。ストリーム型オブジェクトを同時に読み書きする基本クラスを提供します。

メンバ関数

constructor　iopstream (Pstreambuf buf) ;　protected

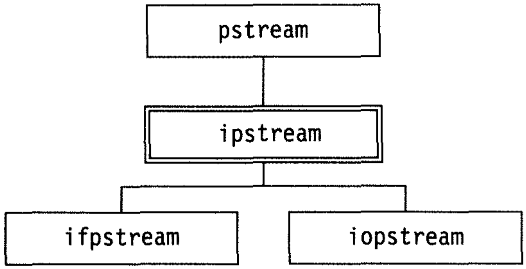
iopstream () ;

最初の形式は，指定されたバッファでバッファリングつきの ***iopstream*** を作成し，*buf* にデータメンバ ***bp*** を設定します。状態は 0 に設定されます。2 番目の形式は各階層のクラスでバッファポインタ ***bp*** を初期化しません。***init*** でバッファと状態を設定してください。

関連項目：***pstream*** : : ***init***

destructor　~iopstream () ;

iopstream オブジェクトを破棄します。



ipstream は，***pstream*** から派生した特殊な入力ストリームです。ストリーム型オブジェクトを読み出す（抽出する）基本クラスです。

メンバ関数

constructor　ipstream (Pstreambuf) ;　　protected

ipstream () ;

最初の形式は，指定されたバッファでバッファリングつきの ***ipstream*** を作成し，*buf* にデータメンバ ***bp*** を設定します。状態は 0 に設定されます。2 番目の形式は各階層のクラスでバッファポインタ ***bp*** を初期化しません。***init*** でバッファと状態を設定してください。

関連項目：***pstream*** : : ***init***

destructor　～ipstream () ;

ipstream オブジェクトを破棄します。

find　PCvoid find (P_id_type Id) ;　　protected

Id に対応するオブジェクトへのポインタを返します。

freadBytes　void freadBytes (void far *data, size_t sz) ;

指定された far バッファ *data* に *sz* で指定されたバイト数分のデータを読み込みます。

freadString char far *freadString () ;

ストリームから文字列を読み出します。文字列の長さを判定して，適切な長さの far 文字配列を確保します。この配列に文字列を読み込み，文字列へのポインタを返します。確保したメモリブロックの解放は呼び出し側で行なわなければなりません。

freadString char far *freadString (char far *buf, unsigned maxLen) ;

指定された far バッファ *buf* にストリームから文字列を読み込みます。文字列の長さが *maxLen* - 1 を超えた場合は何も読み込みません。超えていなければ文字列をバッファに読み込み，ヌルを最後に追加します。

readByte uchar readByte () ;

現在のストリーム位置にある文字を返します。

readBytes void readBytes (Pvoid data, size_t sz) ;

指定されたバッファ *data* に現在のストリーム位置にある *sz* バイトを読み出します。

readData Pvoid readData (PCTStreamableClass, PTStreamable mem) ; protected

読み込むオブジェクトに適した ***read*** 関数を呼び出し，ストリームからデータを読み込みます。*mem* が 0 であれば，該当する ***build*** 関数が最初に呼び出されます。

関連項目：***TStreamableClass***，各ストリームクラスのメンバ関数 ***read*** と ***build***

readPrefix PCTStreamableClass readPrefix () ; protected

現在位置に格納されているクラス ***name*** に対応する ***TStreamableClass*** オブジェクトを返します。

readString Pchar readString () ;

ストリームの現在位置にある文字列のサイズ分のバッファを確保し，それに対し文字列を読み込みます。このバッファは呼び出し側で解放しなければなりません。

readString Pchar readString (Pchar buf, unsigned maxLen) ;

ストリームの現在位置にある文字列を *buf* で指定したバッファに読み込みます。ストリームから読み込む文字列は *maxLen* バイト以内です。

readSuffix void readSuffix () ; protected

オブジェクトの *name* フィールドの最終バイトを読み込んでチェックします。

関連項目：***ipstream*** :: ***readPrefix***

readWord ushort readWord () ;

ストリームの現在位置にあるワードを返します。

registerObject void registerObject (PCvoid adr) ; protected

adr が指すオブジェクトのクラスを登録します。

seekg Ripstream seekg (streampos pos) ;
Ripstream seekg (streamoff off, seek_dir dir) ;

最初の形式は，*pos* で指定された絶対位置にストリーム位置を移動します。2番目の形式は，現在位置を起点として，*dir* からのオフセット *off*（＋または－）分だけ移動します。*dir* は，ストリームの先頭 *beg*，現在位置 *cur*，終端 *end* のいずれかに設定できます。

tellg streampos tellg () ;

ストリームの現在位置（絶対位置）を返します。

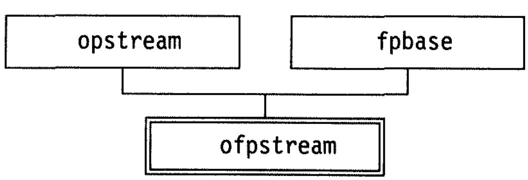
フレンド

operator>>

friend Ripstream operator >> (Ripstream ps, signed char _FAR & ch) ;
friend Ripstream operator >> (Ripstream ps, unsigned char _FAR & ch) ;
friend Ripstream operator >> (Ripstream ps, signed short _FAR & sh) ;
friend Ripstream operator >> (Ripstream ps, unsigned short _FAR & sh) ;
friend Ripstream operator >> (Ripstream ps, signed int _FAR & i) ;
friend Ripstream operator >> (Ripstream ps, unsigned int _FAR & i) ;
friend Ripstream operator >> (Ripstream ps, signed long _FAR & l) ;
friend Ripstream operator >> (Ripstream ps, unsigned long _FAR & l) ;
friend Ripstream operator >> (Ripstream ps, float _FAR & f) ;
friend Ripstream operator >> (Ripstream ps, double _FAR & d) ;
friend Ripstream operator >> (Ripstream ps, long double _FAR & d) ;
friend Ripstream operator >> (Ripstream ps, RTStreamable t) ;
friend Ripstream operator >> (Ripstream ps, RPvoid t) ;

ipstream の *ps* から渡された引数への読み込み（抽出）を行ないます。ストリームへの参照が返されるので，通常どおりに>>操作を連結できます。引数のデータ型により，読み込み方法が決まります。たとえば，符号つきの char の読み込みは ***readByte*** で実現されます。

ofpstream　　OBJSTRM.H



ofpstream は，基本クラス ***fpbase*** と ***opstream*** を単にまとめたものです。ファイルストリームにストリーム型オブジェクトを書き込む（挿入する）基本クラスを提供します。

メンバ関数

constructor ofpstream () ;

ofpstream (PCchar name, int mode = ios : : out,
int prot = filebuf : : openprot) ;

ofpstream (int f) ;

ofpstream (int f, Pchar b, int len) ;

バッファリングつきの ***ofpstream*** オブジェクトを作成します。バッファのサイズと初期内容は，引数 *len* と *b* で設定します。ファイル名，モード，保護の各引数を指定するか，またはファイル記述子 *f* を使えば，ファイルをオープンしてストリームに結びつけることができます。

destructor ～ofpstream () ;

ofpstream オブジェクトを破棄します。

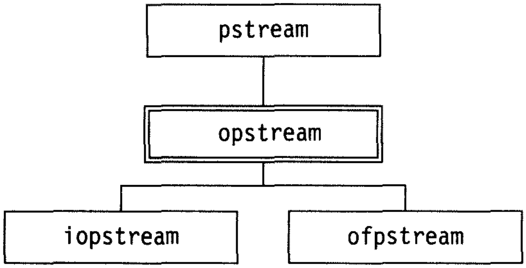
open void open (PCchar name, int mode = ios : : out,
int prot = filebuf : : openprot) ;

指定されたファイルを，指定の *mode*（*app, ate, in, out, binary, trunc, nocreate, noreplace* のいずれか）と保護でオープンします。デフォルトは，*openprot* 保護を持つ *out*（出力）です。オープンしたファイルはこのストリームに結びつけられます。

rdbuf Pfilebuf rdbuf () ;

現在のファイルバッファを返します。

opstream OBJSTRM.H



opstream は，***pstream*** から派生した特殊クラスです。ストリーム型オブジェクトを書き込む（挿入する）基本クラスです。

メンバ関数

constructor opstream (Pstreambuf buf) ;

opstream () ; protected

最初の形式は，指定されたバッファでバッファリングつきの ***opstream*** を作成し，*buf* にデータメンバ ***bp*** を設定します。状態は 0 に設定されます。2 番目の形式は，デフォルトのバッファを確保します。2 番目の形式はバッファポインタ ***bp*** を初期化しません。***init*** でバッファと状態を設定してください。

関連項目：***pstream*** : : ***init***

destructor ~opstream () ;

opstream オブジェクトを破棄します。

find P_id_type find (PCvoid adr) ; protected

adr が指すオブジェクトの型 ID を返します。

flush Ropstream flush () ;

ストリームをフラッシュします。

fwriteBytes void fwriteBytes (const void far *data, size_t sz) ;

指定された far バッファ *data* の指定バイト数 *sz* をストリームに書き込みます。

fwriteString void fwriteString (const char far * str) ;

指定された far 型の文字列 *str* をストリームに書き込みます。

registerObject void registerObject (PCvoid adr) ; protected

adr が指すオブジェクトのクラスを登録します。

seekp Ropstream seekp (streampos pos) ;
Ropstream seekp (streamoff off, seek_dir dir) ;

最初の形式は，*pos* で指定された絶対位置にストリームの現在位置を移動します。2 番目の形式は，現在位置を起点として *dir* からのオフセット *off*（+または -）分だけ移動します。*dir* は，ストリームの先頭 *beg*，現在位置 *cur*，終端 *end* のいずれかに設定できます。

tellp streampos tellp () ;

ストリームの現在位置（絶対位置）を返します。

writeByte void writeByte (uchar ch) ;

ストリームにバイト *ch* を書き込みます。

writeBytes void writeBytes (PCvoid data, size_t sz) ;
data バッファの *sz* バイトをストリームに書き込みます。

writeData void writeData (RTStreamable) ; protected

書き込むオブジェクトに対応するクラスのメンバ関数 ***write*** を呼び出して，ストリームにデータを書き込みます。

関連項目：***TStreamableClass*** とストリームクラスの ***write*** 関数

writePrefix void writePrefix (RCTStreamable) ; protected

クラス名プレフィクスをストリームに書き込みます。<<演算子はこの関数を使って，***writeData*** で書き込まれるデータの前後にプレフィクスとサフィクスを書き込みます。プレフィクスとサフィクスを使うと，ストリームI/Oにおいて型が正しく認識されます。

関連項目：***ipstream*** :: ***readPrefix***

writeString void writeString (PCchar str) ;

str を（長さを示す先行バイトとともに）ストリームに書き込みます。

writeSuffix void writeSuffix (RCTStreamable) ; protected

クラス名サフィクスをストリームに書き込みます。<<演算子はこの関数を使って，***writeData*** で書き込まれたデータの前後にプレフィクスとサフィクスを書き込みます。プレフィクスとサフィクスをつけると，ストリームI/Oにおいて型が正しく認識されます。

関連項目：***ipstream*** :: ***readPrefix***

writeWord void writeWord (ushort us) ;

ワード *us* をストリームに書き込みます。

フレンド

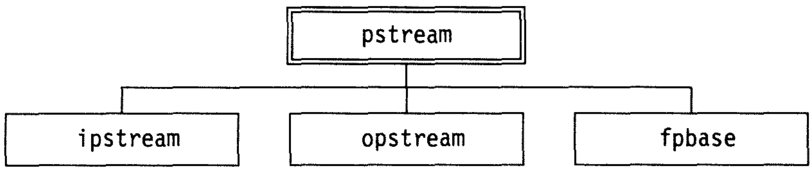
operator<<

```
friend Ropstream operator << (Ropstream ps, signed char ch) ;
friend Ropstream operator << (Ropstream ps, unsigned char ch) ;
friend Ropstream operator << (Ropstream ps, signed short sh) ;
friend Ropstream operator << (Ropstream ps, unsigned short sh) ;
friend Ropstream operator << (Ropstream ps, signed int i) ;
friend Ropstream operator << (Ropstream ps, unsigned int i) ;
friend Ropstream operator << (Ropstream ps, signed long l) ;
friend Ropstream operator << (Ropstream ps, unsigned long l) ;
friend Ropstream operator << (Ropstream ps, float f) ;
friend Ropstream operator << (Ropstream ps, double d) ;
```

friend Ropstream operator << (Ropstream ps, long double d) ;
friend Ropstream operator << (Ropstream ps, RTStreamable t) ;
friend Ropstream operator << (Ropstream ps, PTStreamable t) ;

渡された引数を指定された ***ipstream*** オブジェクトに挿入（書き込み）します。その引数のデータ型により，書き込み操作の形式が決まります。

pstream　　OBJSTRM.H



pstream はストリーム型オブジェクトを扱う基本クラスです。

データメンバ

bp　Pstreambuf bp ;　　**protected**

ストリームバッファへのポインタです。

state　int state ;　　**protected**

書式状態フラグです。***rdstate*** で，現在の状態を調べられます。

関連項目：***pstream*** :: ***rdstate***

types　static PTStreamableTypes types ;　　**protected**

このアプリケーションに登録されているすべての型のデータベースを指すポインタです。

関連項目：***pstream*** :: ***initTypes***

メンバ関数

constructor pstream (Pstreambuf buf) ;

pstream () ; protected

最初の形式は，指定されたバッファを使ってバッファリングつきの ***pstream*** を作成し，***buf*** をデータメンバ ***bp*** に設定します。***state*** は 0 に設定されます。2 番目の形式は，デフォルトのバッファを確保します。

destructor virtual ~pstream () ;

pstream オブジェクトを破棄します。

bad int bad () const ;

エラーが発生すれば，0 以外を返します。

clear void clear (int aState = 0) ;

ストリームの ***state*** に，与えられた値を設定します（デフォルトは 0)。

eof int eof () const ;

ストリームの終端で 0 以外を返します。

error void error (StreamableError, RCTStreamable) ; protected

void error (StreamableError) ;

指定されたエラー状態を設定します。***StreamableError*** は以下のように定義されます。

enum StreamableError { peNotRegistered, peInvalidType } ;

fail int fail () const ;

ストリーム操作が失敗すると 0 以外を返します。

good int good () const ;

state のビットが設定されていなければ，つまりエラーが起こっていなければ，0 以外を返します。

init void init (Pstreambuf sbp) ; protected

ストリームを初期化します。すなわち ***state*** に 0 を，***bp*** に *sbp* を設定します。

initTypes static void initTypes () ;

types が指す，データベースオブジェクトを作成します。***TSteamableClass*** コンストラクタに呼び出されます。

関連項目：***TStreamableClass***

operator void *() operator Pvoid () const ;

void へのポインタ型に型キャストする演算子をオーバーロードします。操作が失敗すると（***pstream*** :: ***fall*** が 0 以外を返すと)，0 を返します。成功すると 0 以外を返します。

関連項目：***pstream*** :: ***fail***

operator!() int operator ! () const ;

NOT 演算子をオーバーロードします。***pstream*** :: ***fail*** が返す値を返します。

関連項目：***pstream*** :: ***fail***

rdbuf Pstreambuf rdbuf () const ;

このストリームに割り当てられたバッファを指す ***pb*** ポインタを返します。

関連項目：***pstream*** :: ***pb***

rdstate int rdstate () const ;

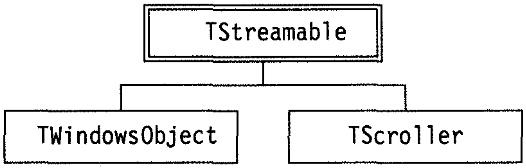
現在の ***state*** 値を返します。

setstate void setstate (int b) ; protected

データメンバ ***state*** を state | = (b & 0xFF)で更新します。

フレンド

プライベートデータベースクラスは ***pstream*** のフレンドです。



TStreamable を継承したクラスはストリームクラスと呼ばれ，オブジェクトをストリームに書き込むことも，ストリームから読み出すこともできます。独自のストリームクラスを開発する場合は，必ず ***TStreamable*** から継承させてください。既存のストリームクラスを基本クラスにするのが一番簡単です。独自のクラスを既存のクラス階層にも組み込まなければならない場合は，多重継承により ***TStreamable*** からクラスを派生させてください。

抽象クラスである ***TStreamable*** からはオブジェクトのインスタンスを構築することはできません。***TStreamble*** から継承したクラスは，3つの純粋仮想関数 ***streamableName, read, write*** を定義し直す必要があります。

メンバ関数

read　virtual Pvoid read (Ripstream) ＝ 0;　　protected

この純粋仮想メンバ関数は各派生クラスで定義し直すか，純粋仮想関数として宣言し直さなければなりません。指定された ***ipstream*** オブジェクトから必要なデータメンバを読み出すように定義し直してください。***read*** をインプリメントするには，通常，基本クラスに ***read*** があればこれを呼び出し，次にこの派生クラスの付加データメンバを抽出します。

関連項目：***TStreamableClass*** :: ***ipstream***

streamalbeName　virtual const Pchar streamableName () const ＝ 0;　　private

この純粋仮想メンバ関数は各派生クラスごとに定義し直すか，純粋仮想関数として宣言し直さなければなりません。この関数を呼び出したオブジェクトのストリームクラス名を返すように定義し直してください。この名前はストリームマネージャがクラスを登録するときに使うので，0で終わる固有の文字列（***TDialog***

など）にしてください。

関連項目：***TStreamableClass, opstream, ipstream***

write virtual void write (Ropstream) = 0; protected

この純粋仮想メンバ関数は各派生クラスごとに定義し直すか，純粋仮想関数として宣言し直さなければなりません，必要なデータメンバを ***opstream*** に書き込むように定義し直してください。***write*** をインプリメントするには，通常，基本クラスの ***write*** があればこれを呼び出して，次に派生クラスの付加データメンバを挿入します。

関連項目：***TStreamableClass, opstream***

フレンド

opstream と ***ipstream*** がフレンドです。

TStreamableClass OBJSTRM.H

```
TStreamableClass
```

ストリームクラスを登録するときに，プライベートデータベースクラスと ***pstream*** によって使われます。

メンバ関数

constructor TStreamableClass (PCchar n, BUILDER b, int d);

指定された名前 *n* とビルダ関数 *b* で ***TStreamable*** オブジェクトを作成し，その型を登録します。各ストリームクラス T*Classname* には *BUILDER* 型のメンバ関数 ***build*** があります（第3章『**ObjectWindows のその他の構成要素**』を参照)。オブジェクトストリーム I/O において型を正しく認識させるため，ストリームマネージャは各クラスの名前と型情報を必要とします。ストリームマネージャにどのアプリケーションにも適切な関数をリンクさせるには，以下のように参照を宣言しなければなりません。

TStreamableClass RegClassName ;

ここで，*TClassName* はストリーム型にするオブジェクトのクラス名です *(RegClassName* は固有の識別子です)。この結果，***TClassName*** が登録され（ストリームマネージャにどの ***build*** 関数を使うかを指示します)，さらに，下位クラスも自動的に登録されます。クラスは何回登録しても，障害にもオーバーヘッドにもなりません。

この関数を呼び出すと，指定したクラスのオブジェクトを読み込むのに必要なメモリが確保されます。ビルド手続きはクラスの専用コンストラクタを呼び出すので，すべての仮想テーブルポインタが正しく初期化されます。

ストリーム型オブジェクトの基底からその ***TStreamable*** 構成要素の先頭まで *d* バイト離れています。*d* は ***__DELTA*** マクロで計算します。たとえば，以下のように行ないます。

```
TStreamableClass RegTClassName = TStreamableClass ("TClassName",
TClassName::build, __DELTA (TClassName)) ;
```

関連項目：***TStreamable,ipstream,opstream***

フレンド

クラス ***opstream*** と ***ipstream*** がフレンドです。

第3章

ObjectWindows のその他の構成要素

この章では，第1章『**ObjectWindows クラス**』と第2章『**ストリームクラス**』で取り上げなかった ObjectWindows の内容を説明します。

この章に記載する型，定数，変数，マクロ，クラス，関数は通常，ObjectWindows のヘッダファイルで宣言します。各項目ごとの一般形式を次の ***Sample*** 例で説明します。

Sample　　Sample のヘッダファイル

宣言　ReturnType Sample (ParamType AParam) ;

Sample は引数 ***AParam*** に対しなんらかの機能を実行します。

関連項目　***Example***

applicationClass 定数　　OWLDEFS.H

説明　***TApplication*** クラスのインスタンスを識別するクラス識別子です。メンバ関数 ***isA*** で返され，コンテナクラスに対してオブジェクトの型を特定します。

関連項目　***TApplication*** : : ***isA***

BF_XXXX 定数　　OWLDEFS.H

説明　以下のボタンフラグ定数が定義されています。

表 3.1
ボタンフラグ定数

定数	値	意味
BF_UNCHEKED	0x0	項目がチェックされていない。
BF_CHECKED	0x1	項目がチェックされている。
BF_GRAYED	0x2	項目は無効。

チェックボックスとラジオボタンは，*BF*_定数で状態を定義します。

関連項目　*TCheckBox* :: *SetCheck, TCheckBox* :: *GetCheck*

BUILDER 型　　OBJSTRM.H

定義　typedef PTStreamable (_FAR *BUILDER)() ;

TStreamableClass コンストラクタへ *build* 関数を渡すときの型として使います。指定された関数はストリームからオブジェクトを読み出す際に，そのオブジェクトのテンプレートを構築するために呼び出されます。

関連項目　*TStreamableClass*

_CLASSDEF（クラス名）マクロ　　_DEFS.H

説明　クラスを定義するときヘッダファイルの先頭で使われます。このマクロはクラス *classname* に対し，以下の typedef を定義します。

```
typedef classname _FAR * Pclassname ;
typedef classname _FAR & Rclassname ;
typedef classname _FAR * _FAR & RPclassname ;
```

```
typedef const classname _FAR * PCclassname ;
typedef const classname _FAR & RCclassname ;
```

以上の typedef は DLL のプログラミングで便利です。***_FAR*** マクロは必要に応じて自動的に“far”に定義されるので，データポインタと参照に対し明示的に使う必要はありません。

_CLASSDLL マクロ

説明 コンパイラのコマンドラインまたは IDE でユーザが指定するマクロです。コンパイルするモジュールが DLL 内でクラスを使うか，または DLL にクラスを渡すかを示します。***_CLASSTYPE***, ***_EXPORT***, ***_FAR*** など他のマクロは，***_CLASSDLL*** が定義されているかどうかを確認します。

_CLASSTYPE マクロ　　_DEFS.H

説明 DLL で使うクラスのヘッダファイルで使います。“huge”, “far”, “near” のいずれか適当なものに展開されます。

__DLL__ または ***_CLASSDLL*** マクロが定義されれば，***_CLASSTYPE*** は “huge” になります。EXE と DLL 間または 2 つの DLL 間でクラスの受け渡しが行われる場合には，ヒュージクラスでなければなりません。***__DLL__*** マクロも ***_CLASSDLL*** マクロも定義されなければ，***_CLASSTYPE*** は以下のように展開されます。

表 3.2 _CLASSTYPE マクロの非 DLL 展開

データモデル	展開形式	ポインタ
タイニィ，スモール，ミディアム	near	near this near vtable
ラージ，コンパクト	far	far this near vtable
ヒュージ	huge	far this far vtable

説明　以下の定数は，ユーザコマンドメッセージと内部コマンドメッセージの開始値またはメッセージ数を定義します。

表 3.3 コマンドメッセージ (CM) 定数

定数	値	意味
CM_COUNT	0x6000	ユーザコマンドメッセージの数。
CM_FIRST	0xA000	ユーザコマンドメッセージの開始値。
CM_INTERNAL	0xFF00	内部予約コマンドメッセージの開始値。
CM_RESERVED	*CM_INTERNAL* - *CM_FIRST*	予約。

以下は，コマンドメッセージを定義する定数識別子とメンバ関数を持つクラスです。

表 3.4　コマンド別の定数

定数	名前	メニューコマンド
CM_EDITCUT	*TEdit::CMEditCut*	Edit | Cut
CM_EDITCOPY	*TEdit::CMEditCopy*	Edit | Copy
CM_EDITPASTE	*TEdit::CMEditPaste*	Edit | Paste
CM_EDITDELETE	*TEdit::CMEditDelete*	Edit | Delete
CM_EDITCLEAR	*TEdit::CMEditClear*	Edit | Clear
DM_EDITUNDO	*TEdit::CMEditUndo*	Edit | Undo
CM_EDITFIND	*TEditWindow::CMEditFind*	Edit | Find
CM_EDITREPLACE	*TEditWindow::CMEditReplace*	Edit | Replace
CM_EDITFINDNEXT	*TEditWindow::CMEditFindNext*	Edit | Find | Next
CM_FILENEW	*TFileWindow::CMFileNew*	File | New
CM_FILEOPEN	*TFileWindow::CMFileOpen*	File | Open
CM_FILESAVE	*TFileWindow::CMFileSave*	File | Save
CM_FILESAVEAS	*TFileWindow::CMFileSaveAs*	File | Save As
CM_ARRANGEICONS	*TMDIFrame::CMArrangeIcons*	Window | Arrange Icons
CM_TILECHILDREN	*TMDIFrame::CMTileChildren*	Window | Tile
CM_CASCADECHILDREN	*TMDIFrame::CMCascadeChildren*	Window | Cascade
CM_CLOSECHILDREN	*TMDIFrame::CMCloseChildren*	Window | Close All
CM_CREATECHILD	*TMDIFrame::CMCreateChild*	Window | Creat eChild
CM_EXIT	*TWindowsObject::CMExit*	

__DELTA マクロ　OBJSTRM.H

説明　ストリーム型オブジェクトの先頭から ***TStreamable*** オブジェクトの構成要素までのオフセットをバイト数で計算するマクロです。引数はストリームクラス名です。

関連項目　***TStreamableClass***

dialogClass 定数　OWLDEFS.H

説明　***TDialog*** クラスのインスタンスを識別するクラス識別子です。メンバ関数 ***isA*** で返され，コンテナクラスに対してオブジェクトの型を特定します。

関連項目　***TDialog*** :: ***isA***

__DLL__ マクロ

説明　-WD スイッチまたは -WDE スイッチを使って DLL モジュールをコンパイルすると自動的に設定されるマクロです。他のマクロや条件コンパイルは，***__DLL__*** を使って DLL が構築されているかどうかを判断します。

EM_XXXX 定数　OWLDEFS.H

説明　以下のエラーコードが定義されています。

表 3.5
エラー状態定数

定数	値	意味
EM_INVALIDCHILD	-1	ウィンドウの子が無効。
EM_INVALIDCLIENT	-2	MDI クライアントウィンドウが作成できなかった。
EM_INVALIDMAINWINDOW	-3	メインウィンドウが作成できなかった。
EM_INVALIDMODULE	-4	Module オブジェクトが設定されなかったのでウィンドウは無効。
EM_INVALIDWINDOW	-5	*Create* が失敗したのでウィンドウは無効。
EM_OUTOFMEMORY	-6	セーフティプールの大きさを超えるメモリ確保がされた。

ObjectWindows が検出したエラー状態を ***EM***_定数で示します。

_EXPORT マクロ　OWLDEFS.H

説明　DLL を構築する（__***DLL***__が設定されている）場合，_***EXPORT*** に展開されます。それ以外は，_***CLASSTYPE*** に展開されます。

_FAR マクロ　_DEFS.H

説明　DLL データとコードのポインタサイズを制御します。これは DLL プログラムが far ポインタを必要とするからです。制御がなければ near（スモールモデルとミディアムモデルの場合）になります。

__DLL__ か _CLASSDLL が定義されていれば，_FAR は far に定義されます。_FAR を用いるのは，引数リスト，戻り値，関数とメンバ関数の far/near 呼び出し，クラスデータメンバそれぞれに対するポインタです。次の表の typedef は，DLL との互換性を要する場合によく使われる型の省略表記です。

表 3.6　DLL 互換の typedef

型	定義	ヘッダファイル	説明
PCchar	typedef const char _FAR * PCchar ;	CLSTYPES.H	定数 char へのポインタ。
Pchar	typedef char _FAR * Pchar ;	CLSTYPES.H	char へのポインタ。
PCvoid	typedef const void _FAR * PCvoid ;	CLSTYPES.H	void 型定数へのポインタ。
Pvoid	typedef void _FAR * Pvoid ;	CLSTYPES.H	void 型へのポインタ
Rint	typedef int _FAR & Rint ;	OWLDEFS.H	int の参照。
RPvoid	typedef void _FAR * _FAR & RPvoid ;	CLSTYPES.H	void 型のポインタの参照。
RTMessage	typedef TMessage _FAR & RTMessage ;	WINDOBJ.H	***TMessage*** 構造体の参照。

GetApplicationObject ()関数　　APPLICAT.H

宣言　extern PTApplication _EXPFUNC GetApplicationObject () ;

説明　アプリケーションまたは DLL から呼び出せます。ObjectWindows アプリケーションの制御下にある場合，この関数は ObjectWindows アプリケーションのアプリケーションオブジェクトを指すポインタ（***PTApplication***）を返します。DDL から呼び出した場合はヌルを返します。

ID_XXXX 定数　　OWLDEFS.H

説明　以下の定数は子 ID メッセージの値または範囲を定義します。

表 3.7
子 ID メッセージ定数

定数	値	意味
ID_COUNT	0x1000	子 ID メッセージの数。
ID_FIRST	0x8000	子 ID メッセージの先頭。
ID_INTERNAL	0x8F00	内部用に予約。
ID_RESERVED	*ID_INTERNAL*—*ID_FIRST*	予約。

以下は，デフォルトの子識別子です。

表 3.8
ID_コマンドオフセット別の値

定数	値	意味
ID_FIRSTCHILD	*ID_RESERVED* + 1	MDI の子 ID。
ID_MDICHILD	*ID_RESERVED* + 2	MDI のクライアントウィンドウ ID。

ObjectWindows は複数の種類の ***ID***_定数を定義しています。それには子識別子の許容範囲を定義する定数と，ObjectWindows がデフォルトとして使う子識別子があります。

__link マクロ　OBJSTRM.H

説明　指定されたクラスの読み込みまたは書き込みに必要なストリーム型オブジェクトサポートをリンカにリンクさせます。引数は ***TStreamableClass*** スタティック登録オブジェクトの名前です（***TClassName*** というクラスの場合は *RegClassName* にすることになっています)。

moduleClass 定数　OWLDEFS.H

説明　***TModule*** クラスのインスタンスを識別するクラス識別子です。メンバ関数 ***isA*** で返され，コンテナクラスに対してオブジェクトの型を特定します。

関連項目　***TModule*** : : ***isA***

NF_XXXX 定数

説明 以下の定数が定義されています。

表 3.9
通知メッセージ定数

定数	値	意味
NF_COUNT	0x1000	通知メッセージの数。
NF_FIRST	0x9000	通知メッセージの先頭。
NF_INTERNAL	0x9F00	内部用で予約されている通知メッセージの先頭。

*NF*_定数は，通知メッセージの位置範囲として 0x9000 から 0x9FFF までを定義します。この範囲のうち 0x9F00 から 0x9FFF までは内部用に予約されています。

演算子 delete

宣言 void operator delete (Pvoid ptr) ;

ObjectWindows はグローバル演算子 ***delete*** を定義し直して，削除の後にセーフティプールをチェックします。セーフティプールが使い尽されているとセーフティプールを確保し直します。この結果，セーフティプールの復元が可能な場合は必ず自動的に復元されます。

関連項目 ***SafetyPool*** : : ***IsExhausted***

演算子 new

宣言 Pvoid operator new (size_t s) ;

ObjectWindows はグローバル演算子 ***new*** を定義し直して，セーフティプールを使うようにします。セーフティプールの確保に失敗すると，***new*** は既存のセーフティプールがあればこれを削除し，再度オブジェクトの確保を試みます。これも失敗すると 0 を返します。

セーフティプールオブジェクトが解放されるとメモリ不足状態になり，この後の ***SafetyPool*** :: ***IsExhausted*** は 1 を返します。

関連項目　***SafetyPool*** :: ***IsExhausted***

OWLGetVersion 関数　　OWL.H

宣言　WORD FAR OWLGetVersion () ;

ObjectWindows ライブラリのバージョン番号を返します。定数 ***OWLVersion*** の値を返します。

OWLVersion 定数　　OWL.H

定義　const int OWLVersion ;

OWLGetVersion 呼び出しで返されたバージョン番号が期待していた値であるかどうかを確認するときに使います。ObjectWindows の DLL を使うときに有効です。

P_id_型　　OBJSTRM.H

定義　typedef unsigned P_id_type ;

ストリーム型オブジェクトを実現するときに，すでに読み書きされたオブジェクトのデータベースを検索する目的で，内部的に使用されます。

SafetyPool クラス

SAFEPOOL.H

SafetyPool

ObjectWindows が使うセーフティプール機構をカプセル化し，アトム操作でインターフェースオブジェクトが構築されるようにします。

データメンバ

safetyPool static Pvoid safetyPool ;

セーフティプールオブジェクトを指すこのポインタは，実行時に *Size* と同じサイズのメモリブロックに設定されます。セーフティプールを使う低メモリ状態を判定するアプリケーションでは，***IsExhausted*** を呼び出すことによって，セーフティプールが使い尽されているかを判定できます。つまり，これでメモリ不足状態かどうかを知ることができます。セーフティプールのサイズを変更するには ***Allocate*** を使います。

関連項目：***SafetyPool :: IsExhausted, SafetyPool :: Size, SafetyPool :: Allocate***

Size static int Size ;

セーフティプールの現在のサイズが入ります。クラスライブラリによって実行時に定数 ***DEFAULT_SAFETY_POOL_SIZE*** の値に初期化されます。

関連項目：***SafetyPool :: Allocate, SafetyPool :: safetyPool***

S Misc

メンバ関数

Allocate static int Allocate (size_t size = DEFAULT_SAFETY_POOL_SIZE) ;

既存のセーフティプールオブジェクトを削除して，指定されたサイズのセーフティプールを確保し直します。サイズの指定がなければ，デフォルトで定数 ***DEFAULT_SAFETY_POOL_SIZE*** の値をとります。成功すると 1 を返し，失敗すると 0 を返します。

IsExhausted static int IsExhausted () ;

このスタティックメンバ関数は，セーフティプールが尽きると 1 を返します。それ以外は 0 を返します。セーフティプールの確保操作によって既存のセーフティプールが開放されると，セーフティプールが尽きたものと見なされ，*safetyPool* に 0 が設定されます。この状態が起こると，***Allocate*** への呼び出しが成功するまで低メモリ状態が続きます。

関連項目：***SafetyPool*** : : ***Allocate***

scrollerClass 定数 OWLDEFS.H

説明 ***TScroller*** クラスのインスタンスを識別するクラス識別子です。メンバ関数 ***isA*** で返され，コンテナクラスに対してオブジェクトの型を特定します。

関連項目 ***TScroller*** : : ***isA***

SD_XXXX 定数 OWLRC.H

説明 以下の定数が定義されています。

表 3.10 標準ダイアログボックス定数

定数	10 進値
SD_FILEOPEN	32512
SD_FILESAVE	32513
SD_INPUTDIALOG	32514
SD_SEARCH	32528
SD_REPLACE	32529

SD_ 定数は ObjectWindows ダイアログボックスの識別子です。***SD_SEARCH*** と ***SD_REPLACE*** のダイアログリソーステンプレートは STDWNDS.DLG にあります。

StreamableInit 型　　OBJSTRM.H

定義　enum StreamableInit {streamableInit} ;

ストリーム型オブジェクトサポートは，ストリームから読み込むオブジェクトに適したクラスのコンストラクタを提供して，オブジェクトを初期化させます。各コンストラクタは引数の1つに列挙型 ***StreamableInit*** を持ちます。

TActionFunc 型　　WINDOBJ.H

定義　typedef void (_FAR * TActionFunc)(Pvoid Child, Pvoid Paramlist) ;

この関数のポインタは ***TWindowsObject*** : : ***ForEach*** に渡されます。

TActionMemFunc 型　　WINDOBJ.H

定義　typedef void (TWindowsObject : :* _FAR TActionMemFunc)(Pvoid Child, Pvoid ParamList) ;

このメンバ関数のポインタは，***TWindowsObject*** : : ***ForEach*** に渡されます。

```
TComboBoxData
```

TComboBox の転送バッファを表わすインターフェースオブジェクトです。

データメンバ

Selection Pchar Selection ;

現在選択されている文字列へのポインタです。文字列はコンボボックスとの間で転送されます。

Strings PArray Strings ;

コンボボックスに転送される文字列の配列です。

メンバ関数

constructor TComboBoxData () ;

TComboBox オブジェクトを構築します。***Strings*** に空の ***Array*** を，***Selection*** にヌルを設定して初期化します。

destructor ~TComboBoxData () ;

Strings と ***Selection*** 用に確保したスペースを削除します。

AddString void AddString (PChar AString, BOOL IsSelected = FALSE) ;

指定された文字列を ***Strings*** の配列に追加します。*IsSelected* が真の場合は，***Selection*** を削除した後に ***AddString*** を ***Selection*** として再割り当てします。

TCondFunc 型　WINDOBJ.H

定義　typedef BOOL (_FAR * TCondFunc)(Pvoid Child, Pvoid Paramlist) ;

この関数のポインタは ***TWindowsObject*** : : ***FirstThat*** に渡されます。

関連項目　***TWindowsObject*** : : ***FirstThat***

TCondMemFunc 型　WINDOBJ.H

定義　typedef BOOL (TWindowsObject : :* _FAR TCondMemFunc)
(Pvoid Child, Pvoid Paramlist) ;

このメンバ関数のポインタは ***TWindowsObject*** : : ***FirstThat*** に渡されます。

関連項目　***TWindowsObject*** : : ***FirstThat***

TDialogAttr 型　DIALOG.H

定義

```
struct _CLASSTYPE TDialogAttr {
    LPSTR Name ;
    DWORD Param ;
  } ;
```

TDialog の作成属性を保持します。データメンバ ***Name*** は，ダイアログリソースの識別子を保持します。また，データメンバ ***Param*** はダイアログボックスの作成時にそのダイアログボックスに渡す引数を保持します。***Param*** は Windows によって受け取られると，***WM_INITDIALOG*** に対応したメッセージ応答関数で使えるようになります。

関連項目　***TDialog*** : : ***Attr***

TF_XXXX 定数 OWLDEFS.H

説明 以下の定数が定義されています。

表 3.11
転送関数定数

定数	値	意味
TF_GETDATA	0	クラスからデータを取得。
TF_SETDATA	1	クラスへデータを送出。
TF_SIZEDATA	2	クラスが転送したデータのサイズを返す。

TF*_**定数はウィンドウクラスのメンバ関数 ***Transfer に渡され，実行すべき操作を指示します。

TListBoxData クラス LISTBOX.H

```
TListBoxData
```

TListBox の転送バッファを表わすインターフェースオブジェクトです。

データメンバ

SelCount int SelCount ;

選択されている項目の数です。

SelStrings PArray SelStrings ;

リストボックスにデータを転送するときに選択される文字列の配列を指すポインタです。逆にリストボックスから転送するとき，***SelStrings*** は現在選択されている文字列を返します。

Strings PArray Strings ;

TListBox へ転送する文字列の配列を指すポインタです。

メンバ関数

constructor TListBoxData () ;

Strings と ***SelStrings*** を構築します。***SelCount*** を 0 に初期化します。

destructor ~TListBoxData () ;

Strings と ***SelStrings*** 用に確保したスペースを削除します。

AddString void AddString (PChar AString, BOOL IsSelected = FALSE) ;

指定された文字列を ***Strings*** に追加します。*IsSelected* が真であれば，文字列を ***SelStrings*** に追加して ***SelCount*** をインクリメントします。

GetSelString void Get SelString (LPSTR Buffer, int BufferSize, int Index = 0) ;

SelStrings 中の *Index* で指定された文字列を検索し，それを *Buffer* にコピーします。*BufferSize* は末尾のヌルも含みます。

GetSelStringLength int GetSelStringLength (int Index = 0) ;

SelStrings 中の *Index* で指定された文字列の長さを返します。ただし，末尾のヌルは含みません。

ResetSelectons void ResetSelections () ;

SelStrings の文字列をすべて除去して，***SelCount*** を 0 に設定します。

SelectString void SelectString (LPSTR AString) ;

AString を ***SelStrings*** に追加し，***SelCount*** をインクリメントします。

定義

```
struct TMessage {
    HWND Receiver ;
    WORD Message ;
    union {
        WORD WParam ;
        struct tagWP {
            BYTE Lo ;
            BYTE Hi ;
        } WP ;
    } ;
    union {
        DWORD LParam ;
        struct tagLP {
            WORD Lo ;
            WORD Hi ;
        } LP ;
    } ;
    long Result ;
} ;
```

Windows からの入力メッセージに関する情報を保持する構造体です。この構造体は ObjectWindows によって書き込まれ，メッセージ応答メンバ関数に渡されます。

Receiver にはメッセージが送られるウィンドウのハンドルが入ります。*Message* にはメッセージ識別子が入ります。メンバ ***WParam*** と ***Lparam*** には Windows から取得したメッセージ固有の情報が入ります。構造体 ***LP*** と ***WP*** は，*WParam* と *LParam* の上位部分と下位部分に容易にアクセスできるように定義されています。*Result* には Windows に返す値が入ります。

TScrollBarData 型　SCROLLBA.H

定義

```
struct TScrollBarData {
    int LowValue;
    int HighValue;
    int Position;
};
```

TScrollBar データを格納する構造体です。*LowValue* と *HighValue* はスクロールバー範囲の設定と取得で使われます。*Position* はスクロールボックス位置の設定と取得で使われます。

関連項目　***TScrollBar*** :: ***Transfer***

TSearchStruct 型　EDITWND.H

定義

```
struct _CLASSTYPE TSearchStruct {
    char SearchText [81] ;
    BOOL CaseSensitive;
    char ReplaceText [81] ;
    BOOL ReplaceAll;
    BOOL PromptOnReplace;
};
```

TSearchStruct は ***TSearchDialog*** から渡された検索/置換用のテキストと関連オプションを保持します。*SearchText* と *ReplaceText* にはユーザが入力した文字列が入ります。他の引数は，ユーザ選択オプションで大小文字の区別および置換方法について指定します。*PromptOnReplace* はユーザに置き換えを確認させるかどうかを示します。

関連項目　***TSearchDialog*** :: ***TSearchDialog,*** ***TEditWindow*** :: ***CMEdtiFind,*** ***TEditWindow*** :: ***CMEditReplace***

TWindowAttr 型　　WINDOW.H

定義

```
struct _CLASSTYPE TWindowAttr {
    DWORD Style;
    DWORD ExStyle;
    int X, Y, W, H;
    LPSTR Menu;
    int Id;
    LPSTR Param;
};
```

TWindow の作成属性を保持します。

Style と *ExStyle* にはウィンドウスタイルと拡張スタイルが入ります。*X* と *Y* にはウィンドウの左上隅の画面座標が入ります。*W* と *H* にはウィンドウの幅と高さが入ります。*Param* には，ウィンドウの作成時に Windows に渡される値が入ります。この値は ***WM_CREATE*** に対応しているメッセージ応答関数で使われます。*Param* は ***TMDIClient*** によって使われます。非 ObjectWindows コードを変換するときに有効です。

Menu にはウィンドウのメニューのリソース識別子が，*Id* にはウィンドウの識別子が入ります。ただし，ダイアログボックスコントロールの場合，*Id* はそのリソース識別子になります。

関連項目　***TWindow*** :: ***Attr***

WB_XXXX 定数　　OWLDEFS.H

説明　以下の値が定義されています。

表 3.12
TWindowsObject 属性マスク

定数	値	対応する属性の意味
WB_ALIAS	0x01	DLL の呼び出し側でウィンドウと対応づけられる。
WB_AUTOCREATE	0x02	親とともに作成される。
WB_FROMRESOURCE	0x04	リソース定義から作成される。
WB_KBHANDLER	0x08	コントロールにキーボードインターフェースを提供する。
WB_MDICHILD	0x10	MDI 子ウィンドウ。
WB_MDIFRAME	0x20	MDI メイン（フレーム）ウィンドウ。
WB_PREDEFINEDCLASS	0x40	定義済みの Windows のクラスがある。
WB_TRANSFER	0x80	***Transfer*** に関与している。

WB_ 定数は，***TWindowsObject*** が内部的に使用するフラグ属性のビットマスクを定義しています。各属性に ***WB_*** マスクが定義されています。

関連項目 ***TWindowsObject* :: *SetFlags*, *TWindowsObject* :: *EnableAutoCreate*, *TWindowsObject* :: *EnableKBHandler*, *TWindowsObject* :: *EnableTransfer***。

windowClass 定数　　OWLDEFS.H

説明 ***TWindow*** クラスのインスタンスを識別するクラス識別子です。メンバ関数 ***isA*** で返され，コンテナクラスに対してオブジェクトの型を特定します。

関連項目 ***TWindows* :: *isA***

WM_XXXX 定数　　OWLDEFS.H

説明　以下の定数が定義されています。

表 3.13
ウィンドウメッセージ定数

定数	**値**	**意味**
WM_COUNT	0x8000	予約メッセージの数。
WM_FIRST	0x0000	予約範囲の先頭。
WM_INTERNAL	0x7F00	ObjectWindows 予約範囲の先頭。
WM_RESERVED = ***WM_INTERNAL*** - ***WM_FIRST***		予約。

ObjectWindows は Windows の ***WM_XXXX*** メッセージに関連する定数を定義します。0x7F00 から 0x7FFF までの範囲が ObjectWindows の内部用に予約されています（第 5 章『**Windows メッセージ**』参照）。

P A R T 2

第2部

Windows API クイックリファレンス

第4章

Windows 関数

この章では，論理的につながりのある関数をグループ別にまとめ，それぞれをアルファベット順の一覧表にしています。***FAR PASCAL*** 型として宣言されているすべての関数は，効率のよい Pascal 呼び出しシーケンスが使えます。

ウィンドウマネージャインターフェースとマクロの一覧表

ウィンドウマネージャインターフェース関数は，メッセージ，ウィンドウ，システム出力のいずれかを処理します。以下の表は，論理的につながりのある関数をグループ別にし，簡略な解説をつけたものです。最後は #define マクロ関数の一覧表です。なお，システムサービスやグラフィックスデバイスインターフェース関数の表はありません。

表 4.1　メッセージ関数

関数	説明
CallWindowProc	指定された関数にメッセージ情報を渡します。
DispatchMessage	指定されたウィンドウのウィンドウ関数にメッセージを渡します。
GetMessage	指定された範囲内のメッセージを取得します。
GetMessagePos	最後のメッセージを取得した時点のマウス位置を取得します。
GetMessageTime	最後のメッセージを取得した時刻を取得します。
InSendMessage	現在のウィンドウが ***SendMessage*** 関数呼び出しによって渡されたメッセージを処理中かどうか調べます。
PeekMessage	アプリケーションキューのメッセージをチェックしてメッセージがあればパラメータで指定された構造体に格納します。
PostAppMessage	アプリケーションにメッセージを送出します。
PostMessage	アプリケーションキューにメッセージを格納します。

表 4.1　メッセージ関数（つづき）

関数	説明
PostQuitMessage	*WM_QUIT* メッセージをアプリケーションに送出します。
ReplyMessage	メッセージに応答します。
SendMessage	ウィンドウにメッセージを送ります。
SetMessageQueue	サイズの違う新しいメッセージキューを作成します。
TranslateAccelerator	メニューコマンドのキーボードアクセラレータを処理します。
TranslateMDISysAccel	MDI 子ウィンドウのコマンドアクセラレータを処理します。
TranslateMessage	仮想キーストロークメッセージを文字メッセージに変換します。
WaitMessage	制御権を他のアプリケーションに渡します。
WinMain	Windows アプリケーショを実行するときのエントリポイントになります。

表 4.2　ウィンドウ作成関数

関数	説明
AdjustWindowRect	指定されたクライアント領域のサイズに基づいてウィンドウのサイズを算出します。
AdjustWindowRectEx	指定されたクライアント領域のサイズに基づいて拡張スタイルのウィンドウサイズを算出します。
CreateWindow	オーバーラップウィンドウ，ポップアップウィンドウ，子ウィンドウを作成します。
CreateWindowEx	拡張スタイルのオーバーラップ，ポップアップ，子の各ウィンドウを作成する。
DefDlgProc	アプリケーションが処理しないダイアログボックスメッセージに対しデフォルト処理を行ないます。
DefFrameProc	アプリケーションが処理しない MDI フレームウィンドウメッセージに対しデフォルト処理を行ないます。
DefMDIChildProc	アプリケーションが処理しない MDI 子ウィンドウメッセージに対しデフォルト処理を行ないます。
DefWindowProc	アプリケーションが処理しないウィンドウメッセージに対しデフォルト処理を行ないます。
DestroyWindow	ウィンドウを破棄します。
GetClassInfo	指定されたウィンドウクラスに関する情報を取得します。
GetClassLong	*WNDCLASS* 構造体からウィンドウクラス情報を取得します。
GetClassName	ウィンドウクラス名を取得します。
GetClassWord	*WNDCLASS* 構造体からウィンドウクラス情報を取得します。
GetLastActivePopup	指定したウィンドウが所有するポップアップウィンドウ群の中でどれが最新のアクティブウィンドウかを判定します。
GetWindowLong	ウィンドウ情報を取得します。
GetWindowWord	ウィンドウ情報を取得します。
RegisterClass	ウィンドウクラスを登録します。
SetClassLong	*WNDCLASS* 構造体の情報を置き換えます。

表 4.2　ウィンドウ作成関数（つづき）

関数	説明
SetClassWord	***WNDCLASS*** 構造体の情報を置き換えます。
SetWindowLong	ウィンドウの属性を変更します。
SetWindowWord	ウィンドウの属性を変更します。
UnregisterClass	ウィンドウクラスをウィンドウクラステーブルから除去します。

表 4.3　表示および移動関数

関数	説明
ArrangeIconicWindows	最小化（アイコン化）した子ウィンドウを再配置します。
BeginDeferWindowPos	***DefWindowPos*** が使うメモリを初期化します。
BringWindowToTop	指定されたウィンドウをウィンドウスタックの一番前に出します。
CloseWindow	指定されたウィンドウを非表示にするか，最小化します。
DeferWindowPos	***EndDeferWindowPos*** で移動またはサイズ変更するためのウィンドウの位置情報を記録します。
EndDeferWindowPos	***DeferWindowPos*** が記録した情報に基づき，複数のウィンドウの配置またはサイズを変更します。
GetClientRect	ウィンドウのクライアント領域の座標を取得します。
GetWindowRect	ウィンドウ全体の寸法を取得します。
GetWindowText	ウィンドウのタイトルを取得します。
GetWindowTextLength	指定されたウィンドウのタイトルまたはコントロール内のテキストの長さを文字数で返します。
IsIconic	ウィンドウがアイコン化されているかどうかを調べます。
IsWindowVisible	指定されたウィンドウが表示されているかどうかを判定します。
IsZoomed	ウィンドウが最大化されているかどうかを判定します。
MoveWindow	ウィンドウのサイズと位置を変更します。
OpenIcon	アイコン化されていたウィンドウを表示します。
SetWindowPos	ウィンドウのサイズ，位置，順序を変更します。
SetWindowText	ウィンドウのタイトルまたはコントロールテキストを設定します。
ShowOwnerPopups	すべてのポップアップウィンドウを表示または非表示にします。
ShowWindow	指定されたウィンドウを表示するか非表示にします。

表 4.4　入力関数

関数	説明
EnableWindow	マウス入力またはキーボード入力を有効または無効にします。
GetActiveWindow	アクティブウィンドウのハンドルを取得します。
GetCapture	マウスキャプチャつきのウィンドウへのハンドルを返します。

表 4.4　入力関数（つづき）

関数	説明
GetCurrentTime	現在の Windows のシステム時間を取得します。
GetDoubleClickTime	マウスのダブルクリック時間を取得します。
GetFocus	現在入力フォーカスを持っているウィンドウのハンドルを取得します。
GetTickCount	システムの起動時からの経過時間を取得します。
IsWindowEnabled	指定されたウィンドウでマウス入力またはキーボード入力が有効かどうかを判定します。
KillTimer	指定されたタイマイベントを取り消します。
ReleaseCapture	マウス入力を解放して通常の入力処理に戻します。
SetActiveWindow	ウィンドウをアクティブにします。
SetCapture	マウス入力を指定されたウィンドウに送るように設定します。
SetDoubleClickTime	マウスのダブルクリック時間を設定します。
SetFocus	指定されたウィンドウに入力フォーカスを割り当てます。
SetSysModalWindow	指定されたウィンドウをシステムモードつきウィンドウにします。
SetTimer	システムタイマイベントを作成します。
SwapMouseButton	マウスボタンの左右の機能を入れ換えます。

表 4.5　ハードウェア関数

関数	説明
EnableHardwareInput	マウス入力またはキーボード入力を有効または無効にします。
GetAsyncKeyState	キーが押されたかどうかを返します。
GetInputState	マウス入力またはキーボード入力があるかどうかを返します。
GetKeyboardState	キーボードのキー状態を保持している配列を取得します。
GetKeyNameText	キーボードドライバリストからキーに割り当てられた名前を取得します。
GetKeyState	仮想キーの状態を取得します。
GetKBCodePage	どの OEM/ANSI テーブルがロードされているかを調べます。
OemKeyScan	0 から 0x0FF までの OEM ASCII コードをスキャンコードとシフト状態に変換します。
SetKeyboardState	キーボードのキー状態テーブルを変更します。
MapVirtualKey	仮想キーコードまたはスキャンコードを受け取り，それを指定されたコードに変換して返します。
VkKeyScan	ANSI 文字を仮想キーコードとシフト状態に変換します。

表 4.6　描画関数

関数	説明
BeginPaint	ウィンドウへの描画用の準備をします。
DrawFocusRect	フォーカスを指示するための矩形を描画します。

表 4.6 描画関数（つづき）

関数	説明
DrawIcon	アイコンを描画します。
DrawText	指定されたテキストを描画します。
EndPaint	ウィンドウの描画の終了を示します。
ExcludeUpdateRgn	ウィンドウの無効領域内での描画を防止します。
FillRect	指定された矩形を指定されたブラシで塗りつぶします。
FrameRect	指定された矩形の周囲に境界線を描画します。
GetDC	クライアント領域のディスプレイコンテキストを取得します。
GetUpdateRect	ウィンドウ領域の矩形領域の寸法を取得します。
GetUpdateRgn	ウィンドウの更新領域を取得します。
GetWindowDC	ウィンドウ全体のディスプレイコンテキストを取得します。
GrayString	淡色表示のテキストを描画します。
InvalidateRect	再描画する矩形を設定します。
InvalidateRgn	再描画する領域を設定します。
InvertRect	指定された矩形の色を反転します。
ReleaseDC	ディスプレイコンテキストを解放します。
UpdateWindow	クライアント領域の再描画をします。
ValidateRect	指定された矩形を再描画領域から削除します。
ValidateRgn	指定された領域を再描画領域から削除します。

表 4.7 ダイアログボックス関数

関数	説明
CheckDlgButton	チェックマークをつけるか除去します。または 3 状態ボタンの状態を変更します。
CheckRadioButton	指定されたボタンをチェックし，他のすべてのチェックマークを除去します。
CreateDialog	モードなしダイアログボックスを作成します。
CreateDialogIndirect	テンプレートからモードなしダイアログボックスを作成します。
CreateDialogIndirectParam	テンプレートからモードなしダイアログボックスを作成し，それにデータを渡します。
CreateDialogParam	モードなしダイアログボックスを作成し，それにデータを渡します。
DefDlgProc	ダイアログボックスに送られる Windows メッセージに対しデフォルト処理を行ないます。
DialogBox	モードつきダイアログボックスを作成します。
DialogBoxIndirect	テンプレートからモードつきダイアログボックスを作成します。
DialogBoxIndirectParam	テンプレートからモードつきダイアログボックスを作成し，それにデータを渡します。
DialogBoxParam	モードつきダイアログボックスを作成し，それにデータを渡します。
DlgDirList	パスと一致したファイル名のリストをリストボックスに格納します。

表 4.7　ダイアログボックス関数（つづき）

関数	説明
DlgDirListComboBox	パスと一致したファイル名をコンボボックスに格納します。
DlgDirSelect	リストボックスで現在選択されている文字列を取得します。
DlgDirSelectComboBox	コンボボックスで現在選択されている文字列を取得します。
EndDialog	モードつきダイアログボックスリソースを終了します。
GetDialogBaseUnits	Windows のダイアログの基本単位を取得します。
GetDlgCtrlID	子ウィンドウの ID 値を返します。
GetDlgItem	ダイアログボックス内にあるコントロールのハンドルを取得します。
GetDlgItemInt	コントロール内のテキストを整数値に変換します。
GetDlgItemText	項目のコントロール内のテキストを取得します。
GetNextDlgGroupItem	グループ内の次または前のコントロールのハンドルを取得します。
GetNextDlgTabItem	次または前の *WS_TABSTOP* を持つコントロールハンドルを取得します。
IsDialogMessage	指定されたダイアログボックスに対するメッセージかどうかを判定します。
IsDlgButtonChecked	ボタンがチェックされているかどうかをテストします。
MapDialogRect	ダイアログボックス座標をクライアント座標に変換します。
SendDlgItemMessage	ダイアログボックス内のコントロールにメッセージを送ります。
SetDlgItemInt	コントロールのタイトルまたはコントロールに整数を表わす文字列を設定します。
SetDlgItemText	コントロールのタイトルまたはコントロールに文字列を設定します。

表 4.8　スクロール関数

関数	説明
GetScrollPos	スクロールボックスの現在位置を取得します。
GetScrollRange	指定されたスクロールバーの最大位置と最小位置を取得します。
ScrollDC	矩形をビット単位で水平方向と垂直方向にスクロールします。
ScrollWindow	クライアント領域の内容を移動します。
SetScrollPos	スクロールボックスを設定します。
SetScrollRange	スクロールバーの最大位置と最小位置を設定します。
ShowScrollBar	スクロールバーを表示または非表示にします。

表 4.9　メニュー関数

関数	説明
AppendMenu	メニューに新しい項目を追加します。
CheckMenuItem	ポップアップメニュー項目の左にチェックマークをつけます。マークがついているときは消します。
CreateMenu	空のメニューを作成します。

表 4.9　メニュー関数（つづき）

関数	説明
CreatePopupMenu	空のポップアップメニューを作成します。
DeleteMenu	メニュー項目を除去し，このメニュー項目に対応したポップアップメニューを削除します。
DestroyMenu	指定されたメニューを削除します。
DrawMenuBar	メニューバーを再描画します。
EnableMenuItem	メニュー項目を有効，無効，淡色表示のいずれかにします。
GetMenu	指定されたウィンドウのメニューのハンドルを取得します。
GetMenuCheckMarkDimensions	メニューのデフォルトチェックマークのビットマップの寸法を取得します。
GetMenuItemCount	メニューの項目数を取得します。
GetMenuItemID	メニュー項目の識別子を取得します。
GetMenuState	メニュー項目の状態を取得します。
GetMenuString	メニュー項目のラベルを取得します。
GetSubMenu	ポップアップメニューのメニューハンドルを取得します。
GetSystemMenu	システムメニューのコピーを作成します。
HiliteMenuItem	最上位レベル（メニューバー）のメニュー項目のハイライト状態と非ハイライト状態を指定します。
InsertMenu	メニュー項目をメニューの指定位置に挿入します。
LoadMenuIndirect	メニューリソースをロードします。
ModifyMenu	メニュー項目を変更します。
RemoveMenu	メニュー項目を削除しますが，ポップアップメニューのハンドルは破棄しません。
SetMenu	ウィンドウの新しいメニューを指定します。
SetMenuItemBitmaps	表示するメニュー項目をビットマップに対応させます。
TrackPopupMenu	ポップアップメニューを表示してメニューの選択項目を監視します。

表 4.10　情報関数

関数	説明
AnyPopup	ポップアップウィンドウが画面上に存在するかどうかを示します。
ChildWindowFromPoint	指定された座標がどの子ウィンドウに含まれるかを判定します。
EnumChildWindows	指定された親ウィンドウに属する子ウィンドウを列挙します。
EnumTaskWindows	指定されたタスクに関連するすべてのウィンドウを列挙します。
EnumWindows	画面上にあるすべてのウィンドウを列挙します。
FindWindow	指定されたクラスとタイトルを持つウィンドウへのハンドルを返します。
GetNextWindow	次または前のウィンドウのハンドルを返します。
GetParent	指定されたウィンドウの親ウィンドウのハンドルを取得します。
GetTopWindow	最上位の子ウィンドウのハンドルを取得します。

表 4.10　情報関数（つづき）

関数	説明
GetWindow	ウィンドウマネージャのリストから目的のハンドルを取得します。
GetWindowTask	ウィンドウに関連しているタスクのハンドルを返します。
IsChild	ウィンドウが指定されたウィンドウの子ウィンドウかどうかを判定します。
IsWindow	ウィンドウが有効であるかどうかを判定します。
SetParent	子ウィンドウの親ウィンドウを変更します。
WindowFromPoint	指定された点座標を含むウィンドウのハンドルを取得します。

表 4.11　システム関数

関数	説明
GetCurrentTime	システムの起動後の経過時間を返します。
GetSysColor	システムカラーを取得します。
GetSystemMetrics	システムメトリックスを取得します。
SetSysColors	システムカラーを設定します。

表 4.12　クリップボード関数

関数	説明
ChangeClipboardChain	クリップボードビューアのチェインからウィンドウを削除します。
CloseClipboard	クリップボードをクローズします。
EmptyClipboard	クリップボードを空にしてクリップボード所有権を付与し直します。
EnumClipboardFormats	有効なクリップボード書式のリストを列挙します。
GetClipboardData	クリップボードのデータを取得します。
GetClipboardFormatName	クリップボード書式名を取得します。
GetClipboardOwner	クリップボードの現在のオーナーであるウィンドウのハンドルを取得します。
GetClipboardViewer	クリップボードビューアチェイン内の最初のウィンドウのハンドルを取得します。
GetPriorityClipboardFormat	クリップボード書式からデータを取得します。
IsClipboardFormatAvailable	指定された書式のデータがあるかどうか調べます。
OpenClipboard	クリップボードをオープンします。
RegisterClipboardFormat	新しいクリップボード書式を登録します。
SetClipboardData	クリップボードのデータハンドルを設定します。
SetClipboardViewer	クリップボードチェインの最初にウィンドウのハンドルを追加します。

表 4.13　エラー関数

関数	説明
FlashWindow	アクティブ状態と非アクティブ状態を交互に繰り返してウィンドウを点滅させます。
MessageBeep	システムスピーカーでビープ音を鳴らします。
MessageBox	指定されたテキストとタイトルでウィンドウを作成します。

表 4.14　キャレット関数

関数	説明
CreateCaret	新しいキャレットを作成します。
DestroyCaret	現在のキャレットを削除します。
GetCaretBlinkTime	キャレットの点滅速度を返します。
GetCaretPos	キャレットの現在位置を返します。
HideCaret	特定のウィンドウでキャレットを非表示にします。
SetCaretBlinkTime	キャレットの点滅速度を設定します。
SetCaretPos	指定された位置にキャレットを移動します。
ShowCaret	新しく作成したキャレットを表示します。または非表示のキャレットを再表示します。

表 4.15　カーソル関数

関数	説明
ClipCursor	カーソルの移動範囲を特定の矩形内に限定します。
CreateCursor	指定したビットパターンからカーソルを作成します。
DestroyCursor	*CreateCursor* で作成したカーソルを削除します。
GetCursorPos	カーソル位置を画面座標で取得します。
LoadCursor	リソースファイルからカーソルをロードします。
SetCursor	カーソルの形状を設定します。
SetCursorPos	カーソルの位置を画面座標で設定します。
ShowCursor	カーソルの表示/非表示を設定します。

表 4.16　フック関数

関数	説明
CallMsgFilter	現在のメッセージフィルタ関数にメッセージとデータを渡します。
DefHookProc	フィルタ関数のチェイン内で次のフィルタ関数を呼び出します。
SetWindowsHook	システムフィルタ関数またはアプリケーションフィルタ関数をインストールします。
UnhookWindowsHook	フィルタ関数チェインから指定したフィルタ関数を削除します。

表 4.17　プロパティ関数

関数	説明
EnumProps	ウィンドウのプロパティリストのエントリを列挙します。
GetProp	ウィンドウのプロパティリストからデータのハンドルを取得します。
RemoveProp	プロパティリストからエントリを除去します。
SetProp	プロパティリストのエントリを変更します。

表 4.18　矩形関数

関数	説明
CopyRect	既存の矩形情報をコピーします。
EqualRect	2つの矩形が同じかどうかを調べます。
InflateRect	指定された矩形を拡大または縮小します。
IntersectRect	2つの矩形の交わる部分の矩形情報を作成します。
OffsetRect	特定の矩形を移動します。
PtInRect	指定された点座標が矩形内にあるかどうかを示します。
SetRectEmpty	空の矩形を作成します。
UnionRect	2つの矩形を含む最小の矩形情報を作成します。

表 4.19
Windows マクロ関数

#define GetBValue (RGB)	#define MAKEINTATOM (int)
#define GetGValue (RGB)	#define MAKELONG (WORD, WORD)
#define GetRValue (RGB)	#define MAKEPOINT (DWORD)
#define HIBYTE (WORD)	#define PALETTERGB (R, G, B)
#define HIWORD (DWORD)	#define PALETTEINDEX (int)
#define LOBYTE (WORD)	#define RGB (R, G, B)
#define LockData (Dummy)	#define UnlockData (Dummy)
#define LOWORD (DWORD)	#define UnlockResource (HANDLE)
#define MAKEINTRESOURCE (int)	

AccessResource

定義 int AccessResource (HANDLE Instance, HANDLE ResInfo) ;

リソースファイルをオープンし，リソースの先頭に配置します。リソースの読み込み後は，ファイルのクローズが必要です。

引数 *Instance*

実行形式ファイルにリソースを含むインスタンスモジュール。

ResInfo

FindResource 関数の呼び出しによって作成されるリソース。

戻り値 DOS のファイルハンドル，リソースがない場合は - 1。

関連項目 ***FindResource***

AddAtom

定義 ATOM AddAtom (LPSTR Str) ;

アトムテーブルに *Str* を加えます。固有文字列インスタンスごとに参照カウントが保持されます。

引数 *Str*

ヌルで終わる文字列。

戻り値 成功の場合は一意なアトムハンドル，失敗の場合は - 1。

関連項目 ***GetAtomName***

AddFontResource

定義 int AddFontResource (LPSTR FileName) ;

FileName フォントリソースファイルのフォントリソースを，システムフォントテーブルに加えます。

引数 *FileName*

ロードされたモジュールまたはヌルで終わる文字列へのハンドル。

戻り値 追加フォント数，追加フォントがない場合は 0。

関連項目 ***WM_FONTCHANGE***

AdjustWindowRect

定義 void AdjustWindowRect (LPRECT Rect, DWORD Style, BOOL Menu) ;

Rect サイズにしたがって，ウィンドウの矩形領域サイズを計算します。メニューがある場合には，単一行メニューとみなされます。

引数 *Rect*

変換するクライアント矩形座標を含む ***RECT*** 構造体。

Style

変換するクライアント矩形を持つウィンドウのスタイル。

Menu

メニューがあるウィンドウの場合は 0 以外。

関連項目 ***CreateWindow***

AdjustWindowRectEx

定義 void AdjustWindowRectEx (LPRECT Rect, DWORD Style, BOOL Menu, DWORD ExStyle) ;

拡張スタイルのウィンドウの矩形サイズを，*Rect* サイズに基づいて計算します。メニューがある場合には，単一行メニューとみなされます。

引数 *Rect*

変換するクライアント矩形座標を含む ***RECT*** 構造体。

Style

変換するクライアント矩形を持つウィンドウのスタイル。

Menu

メニューがあるウィンドウの場合は 0 以外。

ExStyle

作成するウィンドウの拡張スタイル。

関連項目 ***CreateWindowEx***

AllocDStoCSAlias

定義 WORD AllocDStoCSAlias (WORD Selector) ;

コードセグメントセレクタに，*Selector* を割り当てます。

引数 *Selector*

データセグメントセレクタ。

戻り値 成功の場合は対応するコードセグメントセレクタ，失敗の場合は 0。

AllocResource

定義　HANDLE AllocResource (HANDLE Instance, HANDLE ResInfo,
DWORD Size) ;

ResInfo に，初期化されていないメモリを割り当てます。

引数　*Instance*

リソースを含む実行形式ファイルのモジュールインスタンス。

ResInfo

要求するリソース。

Size

リソースに割り当てるバイト数，ただし 0 の場合は無視。

戻り値　割り当てられたグローバルメモリブロック。

関連項目　***FindResource, LoadResource***

AllocSelector

定義　WORD AllocSelector (WORD Selector) ;

Selector のコピーを新セレクタとして割り当てます。*Selector* が NULL の場合には，初期化されていない新セレクタを割り当てます。

➡ 通常，この関数とその関連関数は Windows のプログラミングに不要なので，できるだけ使わないようにします。

引数　*Selector*

コピー対象のセレクタ。

戻り値　成功の場合はセレクタ，失敗の場合は 0。

AnimatePalette

定義 void AnimatePalette (HPALETTE Palette, WORD StartIndex,
WORD NumEntries,
LPPALETTEENTRY PaletteColors) ;

StartIndex と *NumEntries* 間に位置する *Palette* のエントリを，*PaletteColors* で置き換えます。

引数 *Palette*

論理パレット。

StartIndex

パレット内の置換する最初のエントリ。

NumEntries

パレット内の置換するエントリ数。

PaletteColors

PALETTEENTRY 構造体の配列。

関連項目 ***CreatePalette***

AnsiLower

定義 LPSTR AnsiLower (LPSTR Str) ;

言語ドライバによって *Str* を小文字に変換します。

引数 *Str*

ヌルで終わる文字列または下位バイトの1文字。

戻り値 変換された文字列または文字。

AnsiLowerBuff

定義 WORD AnsiLowerBuff (LPSTR Str, WORD Length) ;

言語ドライバによって *Str* を小文字に変換します。

引数 *Str*

文字のバッファ。

Length

バッファ内の文字数で，文字数が 0 の場合のバッファ長は 64 K (65,536)。

戻り値 変換された文字列の長さ。

AnsiNext

定義 LPSTR AnsiNext (LPSTR CurrentChar) ;

複数バイト系の文字列の次の文字を取得します。

引数 *CurrentChar*

ヌルで終わる文字列。

戻り値 文字列内の次の文字のポインタ。

AnsiPrev

定義 LPSTR AnsiPrev (LPSTR Start, LPSTR CurrentChar) ;

複数バイト系の文字列の前の文字を取得します。

引数 *Start*

ヌルで終わる文字列の先頭。

CurrentChar
文字列内の文字のポインタ。

戻り値 文字列内の前の文字のポインタ。

AnsiToOem

定義 int AnsiToOem (LPSTR AnsiStr, LPSTR OemStr) ;

AnsiStr を OEM 定義の文字セットに変換します。変換文字列のバッファ長は 64 K を超えてもかまいません。

引数 *AnsiStr*
ANSI 文字列（ヌルで終わる文字列)。

OemStr
変換文字列のコピー先で，*AnsiStr* と同じでもかまいません。

戻り値 常に -1。

AnsiToOemBuff

定義 void AnsiToOemBuff (LPSTR AnsiStr, LPSTR OemStr, WORD Length) ;

AnsiStr を OEM 定義の文字セットに変換します。

引数 *AnsiStr*
ANSI 文字のバッファ。

OemStr
変換文字列のコピー先で，*AnsiStr* と同じでもかまいません。

Length
AnsiStr のバッファ長で，*AnsiStr* が 0 の場合の長さは 64 K。

AnsiUpper

定義 LPSTR AnsiUpper (LPSTR Str) ;

言語ドライバによって *Str* を大文字に変換します。

引数 *Str*

ヌルで終わる文字列または下位バイトの 1 文字。

戻り値 変換された文字列または文字。

AnsiUpperBuff

定義 WORD AnsiUpperBuff (LPSTR Str, WORD Length) ;

言語ドライバによって *Str* を大文字に変換します。

引数 *Str*

文字のバッファ。

Length

Str のバッファ内の文字数で，*Str* が 0 の場合の長さは 64 K (65,536)。

戻り値 変換された文字列の長さ。

AnyPopup

定義 BOOL AnyPopup ;

画面上のポップアップウィンドウの有無を判定します。

戻り値 ポップアップウィンドウがあれば 0 以外，なければ 0。

AppendMenu

定義 BOOL AppendMenu (HMENU Menu, WORD Flags, WORD IDNewItem,
LPSTR NewItem) ;

Flags で状態を指定した新項目をメニューの最後に追加します。

引数 *Menu*

変更されるメニュー。

Flags

下記の ***MF***_定数のいずれか，または組み合わせ（第7章の「**MF_メニューフラグ**」を参照）。

MF_BITMAP, MF_CHECKED, MF_DISABLED, MF_ENABLED, MF_GRAYED, MF_MENUBARBREAK, MF_MENUBREAK, MF_OWNERDRAW, MF_POPUP, MF_SEPARATOR, MF_STRING, MF_UNCHECKED

IDNewItem

コマンドID，ただしポップアップメニューの場合はメニューハンドル。

NewItem

新しいメニュー文字列，ただし項目にビットマップを使用する場合には，下位ワードにビットマップのハンドルを設定。

戻り値 成功の場合は0以外，失敗の場合は0。

関連項目 ***DrawMenuBar, SetMenuItemBitmaps, WM_DRAWITEM, WM_MEASUREITEM***

Arc

定義 BOOL Arc (HDC DC, int X1, int Y1, int X2, int Y2, int X3, int Y3,
int X4, int Y4) ;

境界矩形の中央に楕円の円弧を描きます。

引数 *DC*

デバイスコンテキスト。

X1, Y1

境界矩形の左上隅。

X2, Y2

境界矩形の右下隅。

X3, Y3

円弧の始点。

X4, Y4

円弧の終点。

戻り値 円弧が描ければ 0 以外，描けなければ 0。

境界矩形の高さと幅は，いずれも 32,767 を超えないものとします。

ArrangeIconicWindows

定義 WORD ArrangeIconicWindows (HWND Wnd) ;

MDI クライアントウィンドウのアイコン，またはデスクトップウィンドウのアイコンを整理します。

引数 *Wnd*

オーナーウィンドウハンドル。

戻り値 1列のアイコンの高さ，アイコンがなければ 0。

関連項目 *GetDesktopWindow*

BeginDeferWindowPos

定義 HANDLE BeginDeferWindowPos (int NumWindows) ;

複数ウィンドウ位置データの構造体にメモリを割り当てます。

引数 *NumWindows*

位置情報を格納するウィンドウの数の初期化。

戻り値 ウィンドウ位置の構造体ハンドル。

関連項目 ***DeferWindowPos, EndDeferWindowPos***

BeginPaint

定義 HDC BeginPaint (HWND Wnd, LPPAINTSTRUCT PaintStruct) ;

WM_PAINT メッセージに答えて，ウィンドウへの描画の用意をします。***Paint*** には描画情報が代入されます。各 ***BeginPaint*** には対応する ***EndPaint*** を呼び出します。

引数 *Wnd*

再描画されるウィンドウ。

Paint

描画情報を受け取る ***PAINTSTRUCT***。

戻り値 デバイスコンテキスト。

関連項目 ***EndPaint, InvalidateRect, InvalidateRgn, WM_ERASEBKGND, WM_PAINT***

BitBlt

定義 BOOL BitBlt (HDC DestDC, int X, int Y, int Width, int Height, HDC SrcDC, int XSrc, int YSrc, DWORD Rop) ;

指定ラスタ操作を行なって，*SrcDC* のビットマップを *DestDC* へコピーします。

引数 *DestDC*

ビットマップ転送先のデバイスコンテキスト。

X, Y

転送先の矩形の左上隅。

Width

転送先の矩形と転送元のビットマップそれぞれの幅。

Height

転送先の矩形と転送元のビットマップそれぞれの高さ。

SrcDC

ビットマップのコピー元のデバイスコンテキスト，ただし *DestDC* のみのラスタ操作の場合は 0。

XSrc, YSrc

SrcDC の左上隅。

Rop

次の 3 次元ラスタ操作定数の 1 つ（第 7 章の「**Ternary Raster Operation**(**3 次元ラスタ操作**)」を参照）。

BLACKNESS, DSTINVERT, MERGECOPY, MERGEPAINT, NOTSRCCOPY, NOTSRCERASE, PATCOPY, PATINVERT, PATPAINT, SRCAND, SRCCOPY, SRCERASE, SRCINVERT, SRCPAINT, WHITENESS

SRCCOPY が行なうのは，転送元から転送先への単純なコピーです。

戻り値 ビットマップが描ければ 0 以外，描けなければ 0。

BringWindowToTop

定義 void BringWindowToTop (HWND Wnd) ;

Wnd をアクティブにして，オーバーラップするウィンドウ群の一番上に重ねます。

引数 *Wnd*
ポップアップまたは子ウィンドウ。

BuildCommDCB

定義 int BuildCommDCB (LPSTR Def, DCB FAR * DCBStruct) ;

Def を *DCB* をコピー先とするデバイス制御ブロックコードに変換します。

引数 *Def*
デバイス制御情報である DOS MODE コマンドライン（ヌルで終わる文字列)。

DCBStruct
格納する ***DCB*** 構造体。

戻り値 *Def* が変換されれば 0, 変換されなければ負の値。

関連項目 ***SetCommState***

CallMsgFilter

定義 BOOL CallMsgFilter (LPMSG Msg, int Code) ;

現在のメッセージフィルタ関数に *Msg* を渡します。

引数 *Msg*

フィルタ対象のメッセージを含む ***MSG*** 構造体。

Code

フィルタ関数コード。

戻り値 メッセージの処理を要する場合は 0, 必要ない場合は 0 以外。

関連項目 ***SetWindowsHook***

CallWindowProc

定義 LONG CallWindowProc (FARPROC PrevWndProc, HWND Wnd,
WORD Msg, WORD wParam, LONG lParam) ;

PrevWndProc を呼び出し，メッセージ情報を渡します。メッセージを受け取り，ウィンドウのサブクラス化を行なってから，そのクラスのウィンドウ関数に渡すことができます。

引数 *PrevWndProc*

以前のウィンドウ関数の手続きインスタンスアドレス。

Wnd

メッセージを受け取るウィンドウ。

Msg

メッセージハンドル。

wParam

追加のメッセージ依存情報。

lParam

追加のメッセージ依存情報。

戻り値 *PrevWndProc* の呼び出しから返される値。

関連項目 ***SetWindowLong***

Catch

定義 int Catch (LPCATCHBUF CatchBuf) ;

システムレジスタのすべての状態と命令ポインタを *CatchBuf* にコピーします。

引数 *CatchBuf*

実行環境のコピー先の ***CATCHBUF***。

戻り値 環境がコピーされれば 0。

関連項目 ***Throw***

ChangeClipboardChain

定義 BOOL ChangeClipboardChain (HWND Wnd, HWND WndNext) ;

クリップボードビューアチェインから *Wnd* を除去して，*WndNext* で置き換えます。

引数 *Wnd*

チェインから除去されるウィンドウ。

WndNext

クリップボードビューアチェインで *Wnd* と置き換わるウィンドウ。

戻り値 ウィンドウが見つかり除去できれば 0 以外。

関連項目 ***SetClipboardViewer***, ***WM_CHANGECBCHAIN***

ChangeMenu

Windows 3.0 ではこの関数のかわりに ***AppendMenu***, ***DeleteMenu***, ***InsertMenu***, ***ModifyMenu***, ***RemoveMenu*** が使われます。

ChangeSelector

定義 WORD ChangeSelector (WORD DestSelector, WORD SourceSelector) ;

セレクタの属性をコードからデータへ，またはこの逆に変換します。セレクタの値は変りませんが，この変換はやむを得ない場合に限ります。また，ソースとセレクタとデスティネーションセレクタの同期をとれないので，セレクタを変換したらただちに使ってください。

引数 *DestSelector*

変換したセレクタを受け取ります。***AllocSelector*** であらかじめ割り当てておいてください。

SourceSelector

変換対象のセレクタ。

戻り値 変換後のセレクタ。変換が失敗すれば，0 を返します。

CheckDlgButton

定義 void CheckDlgButton (HWND Dlg, int IDButton, WORD Check) ;

ボタンコントロールのチェックマークを付加，または除去します。また，3 状態ボタンコントロールの状態の変更を行ないます。

引数 *Dlg*

ボタンのあるダイアログボックス。

IDButton

変更されるボタンコントロール。

Check

除去は 0, チェックは 1, 淡色表示は 2。

CheckMenuItem

定義 BOOL CheckMenuItem (HMENU Menu, WORD IDCheckItem,
WORD Check) ;

ポップアップメニュー項目のチェックマークの付加，または除去を行ないます。

引数 *Menu*

ポップアップメニュー。

IDCheckItem

チェックされるメニュー項目。

Check

項目のチェック方法とチェック位置指定方法。

MF_BYCOMMAND または ***MF_BYPOSITION*** に組み合わせて，***MF_CHECKED*** または ***MF_UNCHECKED*** を使用（第7章の「**MF_メニューフラグ**」を参照)。

戻り値 項目のそれまでの状態，メニュー項目が存在しなければ -1。

CheckRadioButton

定義 void CheckRadioButton (HWND Dlg, int IDFirstButton, int IDLastButton,
int IDCheckButton) ;

IDCheckButton をチェックします。また，*IDFirstButton*，*IDLastButton* が指定するラジオボタングループからチェックマークを除去します。

引数 *Dlg*

ダイアログボックス。

IDFirstButton

グループの最初のラジオボタン ID。

IDLastButton
グループの最後のラジオボタン ID。

IDCheckButton
チェック対象のラジオボタン ID。

ChildWindowFromPoint

定義 HWND ChildWindowFromPoint (HWND WndParent, POINT APoint) ;

APoint を含む子ウィンドウで *WndParent* に属するものを判定します。

引数 *WndParent*
オーナーウィンドウ。

APoint
判定の元となるクライアント座標系の ***POINT*** 構造体。

戻り値 点を含む子ウィンドウ，点がオーナーウィンドウ外にある場合は 0, 点を含む子ウィンドウがなければ *WndParent*。

Chord

定義 BOOL Chord (HDC DC, int X 1, int Y 1, int X 2, int Y 2, int X 3, int Y 3, int X 4, int Y 4) ;

矩形中央の楕円と交わる線分によって弦を描きます。

引数 *DC*
デバイスコンテキスト。

X 1, Y 1
矩形の左上隅。

X 2, Y 2
矩形の右下隅。

X 3, Y 3
線分の一端。

X 4, Y 4
線分の他端。

戻り値 弦が描ければ 0 以外，描けなければ 0。

ClearCommBreak

定義 int ClearCommBreak (int Cid) ;

文字転送の状態に戻すとともに転送ラインのブレーク状態を解除します。

引数 *Cid*
復元される通信デバイス。

戻り値 成功すれば 0，*Cid* (COM ID)が無効デバイスであれば負の値。

関連項目 ***OpenComm***

ClientToScreen

定義 void ClientToScreen (HWND Wnd, LPPOINT Point) ;

クライアント座標を画面座標に変換します。

引数 *Wnd*
クライアント領域を含むウィンドウ。

Point
クライアント座標を含む ***POINT*** 構造体。

ClipCursor

定義 void ClipCursor (LPRECT Rect) ;

カーソル移動を ***Rect*** 内に制限します。*Rect* が NULL であれば，カーソルの移動制限が解除されます。

引数 *Rect*
画面座標系の ***RECT***。

関連項目 ***SetCursorPos***

CloseClipboard

定義 BOOL CloseClipboard () ;

クリップボードをクローズして，他アプリケーションのアクセスを可能にします。

戻り値 クリップボードがクローズされれば 0 以外，クローズされなければ 0。

CloseComm

定義 int CloseComm (int Cid) ;

Cid をクローズするとともに出力キューをフラッシュします。送受信キュー用のメモリすべてが解放されます。

引数 *Cid*
通信デバイス。

戻り値 デバイスがクローズされれば 0, 失敗の場合は負の値。

関連項目 ***OpenComm***

CloseMetaFile

定義 HDC CloseMetaFile (HDC DC) ;

DC をクローズして，メタファイルを再生するハンドルを作成します。

引数 *DC*

メタファイルデバイスコンテキスト。

戻り値 成功すればメタファイルハンドル，失敗の場合は 0。

関連項目 ***PlayMetaFile***

CloseSound

定義 Void CloseSound () ;

音声キューすべてをフラッシュして，その割り当てバッファを解放し，演奏デバイスへのアクセスをクローズします。

CloseWindow

定義 void CloseWindow (HWND Wnd) ;

Wnd を最小化します。オーバーラップウィンドウのアイコンは，画面のアイコン領域に配置されます。

引数 *Wnd*

最小化されるウィンドウ。

CombineRgn

定義 int CombineRgn (HRGN DestRgn, HRGN SrcRgn 1, HRGN SrcRgn 2,
int CombineMode) ;

SrcRgn 1 と *SrcRgn 2* の両領域を結合して，結合した領域を *DestRgn* に置きます。*CombineMode* によって領域の結合方法を指定できます。

引数 *DestRgn*
新領域に置き換えられる領域。

SrcRgn 1
既存の領域。

SrcRgn 2
既存の領域。

CombineMode
RGN_AND, RGN_COPY, RGN_DIFF, RGN_OR, RGN_XOR のいずれかの定数（第7章の「**RGN_領域結合フラグ**」を参照)。

戻り値 ***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION*** のいずれかの定数（第7章の「**Region（領域）フラグ**」を参照)。

CopyMetaFile

定義 HANDLE CopyMetaFile (HANDLE SrcMetaFile, LPSTR FileName) ;

SrcMetaFile を *FileName* ファイルにコピーします。

引数 *SrcMetaFile*
ソースメタファイル。

FileName
ヌルで終わるメタファイル名，ただしメモリメタフィルにコピーする場合は NULL。

戻り値 新しいメタファイルハンドル。

CopyRect

定義 int CopyRect (LPRECT DestRect, LPRECT SourceRect) ;

SourceRect を *DestRect* にコピーします。

引数 *DestRect*

RECT 構造体。

SourceRect

RECT 構造体。

戻り値 現在のところ意味をもちません。

CountClipboardFormats

定義 int CountClipboardFormats () ;

クリップボードによって描画可能な書式数を数えます。

戻り値 クリップボードのデータ書式数。

CountVoiceNotes

定義 int CountVoiceNotes (int Voice) ;

Voice の音符数を数えます。

引数 *Voice*

音声キュー。

戻り値 音符数。

関連項目 ***SetVoiceNote***

CreateBitmap

定義 HBITMAP CreateBitmap (int Width, int Height, BYTE Planes,
BYTE BitCount, LPSTR Bits) ;

デバイス依存のメモリビットマップを作成します。

引数 *Width*

ビットマップの幅（ピクセル単位）。

Height

ビットマップの高さ（ピクセル単位）。

Planes

ビットマップのカラープレーン数。

BitCount

表示ピクセルあたりのカラービット数。

Bits

ビットマップの初期値を含む short int 型配列，ただし NULL の場合の新ビットマップは初期化しません。

戻り値 成功すればビットマップハンドル，失敗すれば NULL。

関連項目 ***BitBlt, SelectObject***

CreateBitmapIndirect

定義 HBITMAP CreateBitmapIndirect (LPBITMAP Bitmap) ;

Bitmap に基づきビットマップを作成します。

引数 *Bitmap*

BITMAP 構造体。

戻り値 成功すればビットマップハンドル，失敗すれば NULL。

関連項目　***BitBlt***

CreateBrushIndirect

定義　HBRUSH CreateBrushIndirect (LPLOGBRUSH LogBrush) ;

LogBrush に基づき論理ブラシを作成します。

引数　*LogBrush*

LOGBRUSH 構造体。

戻り値　成功すれば論理ブラシハンドル，失敗すれば NULL。

CreateCaret

定義　void CreateCaret (HWND Wnd, HBITMAP ABitmap, int Width, int Height) ;

システムキャレットを新たな形状にします。

引数　*Wnd*

新キャレットを所有するウィンドウ。

ABitmap

キャレットを定義するビットマップ。0 の場合は黒のキャレット，1 の場合には淡色表示のキャレット。

Width

キャレットの幅（論理単位)。

Height

キャレットの高さ（論理単位)。

関連項目　***CreateBitmap, CreateDIBitmap, GetSystemMetrics, LoadBitmap***

CreateCompatibleBitmap

定義 HBITMAP CreateCompatibleBitmap (HDC DC, int Width, int Height) ;

DC と互換性のあるビットマップを作成します。

引数 *DC*

デバイスコンテキスト。

Width

ビットマップの幅（ビット単位)。

Height

ビットマップの高さ（ビット単位)。

戻り値 成功すればビットマップハンドル，失敗すれば NULL。

CreateCompatibleDC

定義 HDC CreateCompatibleDC (HDC DC) ;

DC と互換性のあるメモリデバイスコンテキストを作成します。

引数 *DC*

デバイスコンテキスト。NULL であればメモリデバイスコンテキストを作成。

戻り値 成功すれば新メモリデバイスコンテキスト，失敗すれば NULL。

関連項目 ***DeleteDC, GetDeviceCaps***

CreateCursor

定義 HCURSOR CreateCursor (HANDLE Instance, int XHotspot, int YHotspot,
int Width, int Height, LPSTR ANDBitPlane,
LPSTR XORBitPlane) ;

カーソルを作成します。

引数 *Instance*

カーソルを作成するモジュールインスタンス。

Xhotspot, *Yhotspot*

カーソルホットスポットの位置。

Width

カーソルの幅（ピクセル単位)。

Height

カーソルの高さ（ピクセル単位)。

ANDBitPlane

AND マスクを含むバイト配列。

XORBitPlane

XOR マスクを含むバイト配列。

戻り値 成功すればカーソルハンドル，失敗すれば 0。

CreateDC

定義 HDC CreateDC (LPSTR DriverName, LPSTR DeviceName, LPSTR Output,
LPSTR InitData) ;

DriverName のデバイスのコンテキストを作成します。

引数 *DriverName*

デバイスドライバの DOS ファイル名（拡張子がつかず，ヌルで終わる文字列)。

DeviceName

サポート対象の特定デバイス名（ヌルで終わる文字列)。

Output

出力先の DOS ファイル，またはデバイス名（ヌルで終わる文字列)。

InitData

デバイス固有の初期化データを含む ***DEVMODE*** 構造体。

戻り値 成功すればデバイスコンテキストハンドル，失敗すれば NULL。

CreateDialog

定義 HWND CreateDialog (HANDLE Instance, LPSTR TemplateName,
HWND WndParent, FARPROC DialogFunc) ;

TemplateName のダイアログボックスリソースによるモードなしダイアログボックスを作成します。

引数 *Instance*

実行形式ファイルにダイアログボックスリソースを含むモジュールインスタンス。

TemplateName

ダイアログボックスリソース名（ヌルで終わる文字列)。

WndParent

ダイアログボックスを所有するウィンドウ。

DialogFunc
ダイアログ関数の手続きインスタンスアドレス，ただし定義クラスがあれば NULL。

戻り値 成功すればダイアログボックスウィンドウハンドル，失敗すれば0。

関連項目 ***DefDlgProc, MakeProcInstance, WM_INITDIALOG***

CreateDialogIndirect

定義 HWND CreateDialogIndirect (HANDLE Instance, LPSTR Template,
HWND Parent, FARPROC DialogFunc) ;

DialogTemplate に基づきモードなしダイアログボックスを作成します。

引数 *Instance*
モジュールインスタンス。

DialogTemplate
ダイアログボックステンプレートを含む ***DLGTEMPLATE*** 構造体。

WndParent
ダイアログボックスを所有するウィンドウ。

DialogFunc
ダイアログコールバック関数の手続きインスタンスアドレス。

戻り値 成功すればダイアログボックスウィンドウハンドル，失敗すれば NULL。

関連項目 ***DefDlgProc, MakeProcInstance, WM_INITDIALOG***

CreateDialogIndirectParam

定義 HWND CreateDialogIndirectParam (HANDLE Instance, LPSTR Template,
HWND WndParent,
FARPROC DialogFunc,
DWORD InitParam) ;

DialogTemplate に基づきモードなしダイアログボックスを作成します。***CreateDialogIndirect*** と異なり，*InitParam* の引数を指定し，これがコールバック関数に渡されます。

引数 *Instance*

モジュールインスタンス。

DialogTemplate

ダイアログボックステンプレートを含む ***DLGTEMPLATE*** 構造体。

WndParent

ダイアログボックスを所有するウィンドウ。

DialogFunc

ダイアログ関数の手続きインスタンスアドレス，ただし定義クラスがあればNULL。

InitParam

ダイアログボックス作成時に，ダイアログ関数に渡される値（***WM_INITDIALOG*** メッセージの *lParam*）。

戻り値 成功すればダイアログボックスウィンドウハンドル，失敗すれば NULL。

関連項目 ***DefDlgProc, MakeProcInstance, WM_INITDIALOG***

CreateDialogParam

定義 HWND CreateDialogParam (HANDLE Instance, LPSTR TemplateName,
HWND WndParent, FARPROC DialogFunc,
DWORD InitParam) ;

TemplateName に基づきモードなしダイアログボックスを作成します。

引数 *Instance*

実行形式ファイルにダイアログボックスを含むモジュールインスタンス。

TemplateName

ダイアログボックステンプレート名（ヌルで終わる文字列）。

Parent

ダイアログボックスを所有するウィンドウ。

DialogFunc

ダイアログ関数の手続きインスタンスアドレス，ただし定義クラスがあればNULL。

InitParam

ダイアログボックス作成時に，ダイアログ関数に渡される値（***WM_INITDIALOG*** メッセージの *lParam*）。

戻り値 成功すればダイアログボックスウィンドウハンドル，失敗すれば NULL。

関連項目 ***DefDlgProc, MakeProcInstance, WM_INITDIALOG***

CreateDIBitmap

定義 HBITMAP CreateDIBitmap (HDC DC, LPBITMAPINFOHEADER InfoHeader,
DWORD InitParam, LPSTR InitBits,
LPBITMAPINFO InitInfo, WORD Usage) ;

InfoHeader と *InitInfo* に記述されるデバイス独立のビットマップから，デバイス固有のメモリビットマップを作成します。

引数 *DC*

デバイスコンテキスト。

InfoHeader

ビットマップサイズと書式を記述する ***BITMAPINFOHEADER***。

InitParam

CBM_INIT であれば，*InitBits* と *InitInfo* にしたがってビットマップを初期化。

InitBits

InitInfo の *biBitCount* フィールドによって決まる書式のビットマップの初期値を含むバイト配列。

InitInfo

寸法とカラー書式を記述する ***BITMAPINFO*** 構造体。

Usage

DIB_RGB_COLORS または ***DIB_PAL_COLORS*** のいずれかの定数（第7章の「**DIB_カラーテーブルハンドル**」を参照)。

戻り値 成功すればビットマップハンドル，失敗すれば NULL。

CreateDIBPatternBrush

定義 HBRUSH CreateDIBPatternBrush (GLOBALHANDLE PackedDIB,
WORD Usage) ;

PackedDIB で定義されたデバイス独立ビットマップから，論理ブラシを作成します。

引数 *PackedDIB*

BITMAPINFO 構造体とピクセルの配列を含むグローバルメモリ。

Usage

DIB_RGB_COLORS または ***DIB_PAL_COLORS***（第7章の「**DIB_カラーテーブルハンドル**」を参照）。

戻り値 成功すればブラシハンドル，失敗すればNULL。

関連項目 ***GetDeviceCaps***

CreateDiscardableBitmap

定義 HBITMAP CreateDiscardableBitmap (HDC DC, int Width, int Height) ;

DC と互換性のある廃棄可能ビットマップを作成します。

引数 *DC*

デバイスコンテキスト。

Width

ビットマップの幅（ビット単位）。

Height

ビットマップの高さ（ビット単位）。

戻り値 成功すればビットマップハンドル，失敗すればNULL。

CreateEllipticRgn

定義　HRGN CreateEllipticRgn (int X 1, int Y 1, int X 2, int Y 2) ;

指定矩形内に楕円形の領域を作成します。

引数　*X 1, Y 1*

矩形の左上隅。

X 2, Y 2

矩形の右下隅。

戻り値　成功すれば新領域ハンドル，失敗すれば NULL。

CreateEllipticRgnIndirect

定義　HRGN CreateEllipticRgnIndirect (LPRECT Rect) ;

Rect の指定矩形内に楕円形の領域を作成します。

引数　*Rect*

矩形の左上隅と右下隅を含む ***RECT*** 構造体。

戻り値　成功すれば新領域ハンドル，失敗すれば NULL。

CreateFont

定義　HFONT CreateFont (int Height, int Width, int Escapement, int Orientation,
int Weight, BYTE Italic, BYTE Underline,
BYTE StrikeOut, BYTE CharSet,
BYTE OutputPrecision, BYTE ClipPrecision,
BYTE Quality, BYTE PitchAndFamily,
LPSTR FaceName) ;

GDIの物理フォントの中から指定特性に合うものを選択して，その論理フォントを作成します。

引数 *Height*

フォントの高さ（論理単位）。

Width

平均フォント幅（論理単位）。

Escapement

ライン角度（1/10度単位）。

Orientation

文字ベースライン角度（1/10度単位）。

Weight

0から1000までのフォントの太さ，または***FW_BOLD, FW_NORMAL***などの***FW_***定数のいずれか（第7章の「**FW_フォントウェイトフラグ**」を参照）。

Italic

イタリックフォント。

Underline

下線つきフォント。

StrikeOut

ストライクアウトつきフォント。

CharSet

ANSI_CHARSET, OEM_CHARSET, SYMBOL_CHARSET, SHIFTJIS_CHARSET, HANGEUL_CHARSETのいずれかの定数。

OutputPrecision

OUT_CHARACTER_PRECIS, OUT_DEFAULT_PRECIS, OUT_STRING_PRECIS, OUT_STROKE_PRECISのいずれかの定数。

ClipPrecision

CLIP_CHARACTER_PRECIS, CLIP_DEFAULT_PRECIS, CLIP_STROKE_PRECISのいずれかの定数。

Quality
DEFAULT_QUALITY, DRAFT_QUALITY, PROOF_QUALITY のいずれかの定数。(第7章の「**FF_フォントファミリーフラグ**」を参照)。

PitchAndFamily
DEFAULT_PITCH, FIXED_PITCH, VARIABLE_PITCH のいずれかの定数と ***FF_DECORATIVE, FF_DONTCARE, FF_MODERN, FF_ROMAN, FF_SCRIPT, FF_SWISS*** のいずれかの組み合わせ(第7章の「**FF_フォントファミリーフラグ**」を参照)。

FaceName
フォント名(ヌルで終わる文字列)。

戻り値 成功すれば論理フォントハンドル，失敗すれば0。

関連項目 ***EnumFonts***

CreateFontIndirect

定義 HFONT CreateFontIndirect (LPLOGFONT LogFont) ;

ALogFont で指定した特性に基づき，GDIの物理フォントの中から合うものを選択して，その論理フォントを作成します。

引数 *ALogFont*
LOGFONT 構造体。

戻り値 成功すれば論理フォントハンドル，失敗すれば NULL。

CreateHatchBrush

定義 HBRUSH CreateHatchBrush (int Index, COLORREF Color) ;

指定ハッチスタイルの論理ブラシを作成します。

引数 *Index*

HS_BDIAGONAL, HS_CROSS, HS_DIAGCROSS, HS_FDIAGONAL, HS_HORIZONTAL, HS_VERTICALのいずれかの定数（第7章の「**HS_ハッチスタイル**」を参照）。

Color

ハッチの色を指定する ***COLORREF***。

戻り値 成功すれば論理ブラシハンドル，失敗すれば NULL。

CreateIC

定義 HDC CreateIC (LPSTR DriverName, LPSTR DeviceName, LPSTR Output, LPSTR InitData) ;

デバイスの情報コンテキストを作成します。

引数 *DriverName*

デバイスドライバの DOS ファイル名（拡張子がなくヌルで終わる文字列）。

DeviceName

特定デバイス名（ヌルで終わる文字列）。

Output

出力先の DOS ファイルまたはデバイス名（ヌルで終わる文字列）。

InitData

デバイス固有の初期化データ，ただしデフォルトの初期化には NULL。

戻り値 成功すれば情報コンテキストハンドル，失敗すれば NULL。

CreateIcon

定義 HICON CreateIcon (HANDLE Instance, int Width, int Height, BYTE Planes, BYTE BitsPixel, LPSTR ANDbits, LPSTR XORbits) ;

アイコンを作成します。

引数 *Instance*

アイコンを作成するモジュールインスタンス。

Width

アイコンの幅（ピクセル単位)。

Height

アイコンの高さ（ピクセル単位)。

Planes

XOR マスクのプレーン数。

BitsPixel

XOR マスクのピクセルあたりのビット数。

ANDbits

アイコンのモノクロ用 AND マスクを含むバイト配列。

XORbits

XOR マスクを含むバイト配列。

戻り値 成功すればアイコンハンドル，失敗すれば NULL。

CreateMenu

定義 HMENU CreateMenu () ;

空の初期メニューを作成します。

戻り値 成功すればメニューハンドル，失敗すれば NULL。

関連項目 ***AppendMenu***，***InsertMenu***

CreateMetaFile

定義 HANDLE CreateMetaFile (LPSTR FileName) ;

メタファイルデバイスコンテキストを作成します。

引数 *FileName*

メタファイル名（ヌルで終わる文字列），ただしメモリメタファイルを指定する場合は NULL。

戻り値 成功すればメタファイルデバイスコンテキストハンドル，失敗すれば NULL。

CreatePalette

定義 HPALETTE CreatePalette (LPLOGPALETTE LogPalette) ;

論理カラーパレットを作成します。

引数 *LogPalette*

論理パレットに関するカラー情報を含む ***LOGPALETTE*** 構造体。

戻り値 成功すれば論理パレットハンドル，失敗すれば NULL。

CreatePatternBrush

定義 HBRUSH CreatePatternBrush (HBITMAP Bitmap) ;

Bitmap パターンの論理ブラシを作成します。

引数 *Bitmap*

HBITMAP ビットマップハンドル。

戻り値 成功すれば論理ブラシハンドル，失敗すれば NULL。

関連項目 ***CreateBitmap, CreateBitmapIndirect, LoadBitmap, CreateCompatibleBitmap, DeleteObject, GetDeviceCaps***

CreatePen

定義 HPEN CreatePen (int PenStyle, int Width, COLORREF Color) ;

論理ペンを作成します。

引数 *PenStyle*

PS_SOLID, PS_DASH, PS_DOT, PS_DASHDOT, PS_DASHDOTDOT, PS_NULL, PS_INSIDEFRAME のいずれかの定数（第 7 章の「**PS_ペンスタイル**」を参照）。

Width

論理単位のペン幅。

Color

ペンの ***COLORREF***。

戻り値 成功すれば論理ペンハンドル，失敗すれば NULL。

CreatePenIndirect

定義 HPEN CreatePenIndirect (LOGPEN FAR *LogPen) ;

LogPen に基づき論理ペンを作成します。

引数 *LogPen*

LOGPEN 構造体。

戻り値 成功すれば論理ペンハンドル，失敗すれば NULL。

CreatePolygonRgn

定義 HRGN CreatePolygonRgn (LPPOINT Points, int Count, int PolyFillMode) ;

多角形領域を作成します。

引数 *Points*

多角形の頂点を含む ***POINT*** 構造体の配列。

Count

Points の頂点の数。

PolyFillMode

領域を塗りつぶすための ***ALTERNATE*** または ***WINDING*** 定数（第 7 章の「**PolyFill モード**」を参照）。

戻り値 成功すれば新領域ハンドル，失敗すれば NULL。

関連項目 ***SetPolyFillMode***

CreatePolyPolygonRgn

定義 HRGN CreatePolyPolygonRgn (LPPOINT Points, LPINT PolyCounts, int Counts, int PolyFillMode) ;

相互にオーバーラップする場合もある複数の閉鎖多角形から成る領域を作成します。

引数 *Points*

多角形の頂点を含む ***POINT*** 構造体の配列。

PolyCounts

Points における各多角形の頂点の数を表わす整数配列。

Count

PolyCounts のサイズ。

PolyFillMode

ALTERNATE または ***WINDING*** (第 7 章の「**PolyFill モード**」を参照)。

戻り値 成功すれば領域ハンドル, 失敗すれば NULL。

CreatePopupMenu

定義 HMENU CreatePopupMenu () ;

空のポップアップメニューを作成します。

戻り値 成功すればメニューハンドル, 失敗すれば NULL。

関連項目 ***AppendMenu, InsertMenu, TrackPopupMenu***

CreateRectRgn

定義 HRGN CreateRectRgn (int X 1, int Y 1, int X 2, int Y 2) ;

指定矩形枠内に矩形領域を作成します。

引数 *X 1, Y 1*

矩形枠の左上隅。

X 2, Y 2

矩形枠の右下隅。

戻り値 成功すれば領域ハンドル，失敗すれば NULL。

CreateRectRgnIndirect

定義 HRGN CreateRectRgnIndirect (LPRECT Rect) ;

Rect 枠内に矩形領域を作成します。

引数 *Rect*

領域の左上隅と右下隅を含む ***RECT*** 構造体。

CreateRoundRectRgn

定義　HRGN CreateRoundRectRgn (int X 1, int Y 1, int X 2, int Y 2, int X 3, int Y 3) ;

指定領域の枠内に，角の丸い矩形領域を作成します。

引数　*X 1, Y 1*

領域の左上隅。

X 2, Y 2

領域の右下隅。

X 3

丸角部分の楕円の幅。

Y 3

丸角部分の楕円の高さ。

戻り値　成功すれば領域ハンドル，失敗すれば NULL。

CreateSolidBrush

定義　HBRUSH CreateSolidBrush (COLORREF Color) ;

論理ブラシを作成します。

引数　*Color*

ブラシの ***COLORREF***。

戻り値　成功すれば論理ブラシハンドル，失敗すれば NULL。

CreateWindow

定義 HWND CreateWindow (LPSTR ClassName, LPSTR WindowName,
DWORD Style, int X, int Y, int Width, int Height,
HWND WndParent, HMENU Menu,
HANDLE Instance, LPSTR Param) ;

オーバーラップウィンドウ，ポップアップウィンドウ，子ウィンドウを作成します。

引数 *ClassName*

ウィンドウクラス名（ヌルで終わる文字列），または定義済みのコントロールクラス名。

WindowName

ウィンドウのキャプションまたは名称（ヌルで終わる文字列）。

Style

DS _, WS _, BS _, CBS _, ES _, LBS _, SBS _, SS _などのウィンドウまたはコントロールスタイル定数のいずれか，または，その組み合わせ(第7章を参照)。

X, Y

初期ウィンドウ位置，または ***CW _USEDEFAULT***（第7章の「**CW_USEDEFAULT 定数**」を参照)。

Width

初期ウィンドウの幅（デバイス単位)。

Height

初期ウィンドウの高さ（デバイス単位)。

WndParent

オーナーウィンドウ。

Menu

メニューまたは子ウィンドウハンドル。

Instance

関連するモジュールインスタンス。

Param

WM_CREATE メッセージの *lParam* の ***CREATESTRUCT*** に渡される値で，MDI クライアントウィンドウを作成する場合は ***CLIENTCREATESTRUCT*** へのポインタであることが条件。

戻り値 成功すればウィンドウハンドル，失敗すれば NULL。

関連項目 ***RegisterClass, WM_CREATE, WM_GETMINMAXINFO, WM_NCCREATE***

CreateWindowEx

定義

```
HWND CreateWindowEx (DWORD ExStyle, LPSTR ClassName,
                     LPSTR WindowName,
                     DWORD Style, int X, int Y,int Width, int Height,
                     HWND WndParent, HMENU Menu,
                     HANDLE Instance, LPSTR Param) ;
```

拡張スタイルのオーバーラップウィンドウ，ポップアップウィンドウ，子ウィンドウを作成します。

引数 *ExStyle*

WS_EX_DLGMODALFRAME または ***WS_EX_NOPARENTNOTIFY*** のいずれかの拡張ウィンドウスタイル（第 7 章の「**WS_EX_拡張ウィンドウスタイル**」を参照）。

ClassName

ヌルで終わる文字列のウィンドウクラス名，または定義済みのコントロールクラス名。

WindowName

ウィンドウのキャプションまたは名称（ヌルで終わる文字列)。

Style

DS_, WS_, BS_, CBS_, ES_, LBS_, SBS_, SS_ などのウィンドウまたはコントロールスタイル定数のいずれか，または，その組み合わせ(第 7 章を参照)。

X, Y

初期ウィンドウ位置，または ***CW_USEDEFAULT***（第7章の「**CW_USEDEFAULT 定数**」を参照)。

Width

初期ウィンドウの幅（デバイス単位)。

Height

初期ウィンドウの高さ（デバイス単位)。

WndParent

オーナーウィンドウ。

Menu

メニューまたは子ウィンドウハンドル。

Instance

関連するモジュールインスタンス。

Param

WM_CREATE メッセージの *lParam* の ***CREATESTRUCT*** に渡される値で，MDI クライアントウィンドウ作成の場合は，***CLIENTCREATESTRUCT*** のポインタであることが条件。

戻り値 成功すればウィンドウハンドル，失敗すれば NULL。

関連項目 ***CreateWindow, WM_PARENTNOTIFY***

DebugBreak

定義 void DebugBreak() ;

デバッガにブレークをかけます。

DefDlgProc

定義 DWORD DefDlgProc (HWND Dlg, WORD Msg, WORD wParam, DWORD lParam) ;

プライベートウィンドウクラスのダイアログに対しデフォルト処理を行ないます。

引数 *Dlg*

ダイアログボックスハンドル。

Msg

メッセージ番号。

wParam

メッセージ依存情報。

lParam

メッセージ依存情報。

戻り値 メッセージ処理の結果。

DeferWindowPos

定義 HANDLE DeferWindowPos (HANDLE WinPosInfo, HWND Wnd, HWND WndInsertAfter, int X, int Y, int cX, int cY, WORD Flags) ;

Wnd で指定したウィンドウの *WinPosInfo* を更新します。

引数 *WinPosInfo*

複数のウィンドウ位置データの構造体のハンドル。

Wnd

更新情報を格納するウィンドウ。

WndInsertAfter

Wnd 挿入位置の後のウィンドウ。

X, Y

ウィンドウの左上隅の位置。

cX

ウィンドウの新しい幅。

cY

ウィンドウの新しい高さ。

Flags

SWP_DRAWFRAME, SWP_HIDEWINDOW, SWP_NOACTIVATE, SWP_NOMOVE, SWP_NOSIZE, SWP_NOREDRAW, SWP_NOZORDER, SWP_SHOWWINDOW のいずれかのフラグ（第7章の「**SWP_ウィンドウ位置設定フラグ**」を参照）。

戻り値 更新された複数のウィンドウ位置データの構造体。

関連項目 ***BeginDeferWindowPos, EndDeferWindowPos***

DefFrameProc

定義 DWORD DefFrameProc (HWND Wnd, HWND MDIClient, WORD Msg,
WORD wParam, DWORD lParam) ;

MDI フレームウィンドウのデフォルトメッセージ処理を行ないます。

引数 *Wnd*

MDI フレームウィンドウ。

MDIClient

MDI クライアントウィンドウ。

Msg

メッセージ番号。

wParam

メッセージ依存情報。

lParam

メッセージ依存情報。

戻り値　メッセージ処理の結果。

DefineHandleTable

定義　BOOL DefineHandleTable (WORD Offset) ;

アプリケーションのデフォルトデータセグメントにプライベートハンドルテーブルを作成します。このテーブルには，***GlobalLock*** から返されロックされたグローバルメモリオブジェクトのセグメントアドレスが保持されます。リアルモードで実行すると，メモリオブジェクトが移動したときに，そのアドレスをWindowsが更新します。なお，標準モードや386エンハンスモードはプロテクトモードなので，アドレスの更新はありません。

ハンドルテーブルは2つの ***WORD*** 型の値とアドレスにより構成されます。最初のワードはテーブルの項目数です。これは ***DefineHandleTable*** を呼び出す前に初期化してください。2番目のワードはWindowsがLRUメモリのリストを更新した時に0に設定される項目数です。どちらのワードも随時変更できます。***GlobalLock*** はテーブルにあるアドレスを返します。

引数　*Offset*

データセグメントにおけるテーブルのオフセット。その値が0であれば，テーブルの更新がもう必要ないことを示します。

戻り値　成功すれば0以外，失敗すれば0を返します。

DefHookProc

定義　DWORD DefHookProc (int Code, WORD wParam, DWORD lParam,
FARPROC NextHook) ;

フック（メッセージフィルタ）関数チェイン内の次の関数を呼び出します。

引数 *Code*

メッセージ処理方法を指定。

wParam

メッセージ依存情報。

lParam

メッセージ依存情報。

NextHook

次のフック関数の ***FARPROC***。

戻り値 コードに依存する値。

関連項目 ***SetWindowsHook***, ***UnhookWindowsHook***

DefMDIChildProc

定義 DWORD DefMDIChildProc (HWND Wnd, WORD Msg, WORD wParam, DWORD lParam) ;

MDI 子ウィンドウのデフォルトメッセージ処理を行ないます。

引数 *Wnd*

MDI 子ウィンドウ。

Msg

メッセージ番号。

wParam

メッセージ依存情報。

lParam

メッセージ依存情報。

戻り値 メッセージ処理の結果。

DefWindowProc

定義 DWORD DefWindowProc (HWND Wnd, WORD Msg, WORD wParam, DWORD lParam) ;

アプリケーションによる明示的な処理対象とならないメッセージの，デフォルト処理を行ないます。

引数 *Wnd*

ウィンドウハンドル。

Msg

メッセージ番号。

wParam

メッセージ依存情報。

lParam

メッセージ依存情報。

戻り値 メッセージ処理の結果。

DeleteAtom

定義 ATOM DeleteAtom (ATOM AnAtom) ;

アトムを削除します。アトムの参照カウントが0になったら，対応文字列はアトムテーブルから除去されます。

引数 *AnAtom*

アトムハンドル。

戻り値 成功すればNULL，失敗すればアトムハンドル。

DeleteDC

定義 BOOL DeleteDC (HDC DC) ;

デバイスコンテキストを削除します。*DC* がデバイスの最終コンテキストの場合は，デバイスに通知してからすべてのメモリ領域とシステムリソースを解放します。

引数 *DC*
デバイスコンテキストハンドル。

戻り値 成功すれば 0 以外，失敗すれば 0。

DeleteMenu

定義 BOOL DeleteMenu (HMENU Menu, WORD Position, WORD Flags) ;

メニューから 1 つの項目を削除します。その項目がポップアップの場合は，ハンドルが破棄されてメモリが解放されます。

引数 *Menu*
メニューハンドル。

Position
位置またはコマンド ID。

Flags
MF_BYPOSITION, ***MF_BYCOMMAND*** のいずれかのメニュー定数（第 7 章の「**MF_メニューフラグ**」を参照)。

戻り値 成功すれば 0 以外，失敗すれば 0。

DeleteMetaFile

定義 BOOL DeleteMetaFile (HANDLE MF) ;

メタファイルハンドルを無効にして，対応するシステムリソースを解放します。メタファイル自体は削除されません。

引数 *MF*
メタファイルハンドル。

戻り値 成功すれば 0 以外，*MF* が無効ハンドルであれば 0。

関連項目 ***GetMetaFile***

DeleteObject

定義 BOOL DeleteObject (HANDLE Handle) ;

メモリから ***Handle*** を削除して，対応するシステムリソースをすべて解放します。

引数 *Handle*
論理ペン，ブラシ，フォント，ビットマップ，領域，パレットへの各ハンドル。

戻り値 削除されれば 0 以外，*Handle* が無効またはデバイスコンテキストに現在選択されている場合は 0。

DestroyCaret

定義 void DestroyCaret () ;

現在のキャレットを破棄してオーナーウィンドウから解放し，さらに画面に表示があれば，それも除去します。

DestroyCursor

定義 BOOL DestroyCursor (HCURSOR Cursor) ;

Cursor を破棄して対応するメモリを解放します。

引数 *Cursor*

カーソルハンドル。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***CreateCursor***

DestroyIcon

定義 BOOL DestroyIcon (HICON Icon) ;

アイコンを破棄して対応するメモリを解放します。

引数 *Icon*

アイコンハンドル。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***CreateIcon***

DestroyMenu

定義 BOOL DestroyMenu (HMENU Menu) ;

メニューを破棄して対応するメモリを解放します。

引数 *Menu*

メニューハンドル。

戻り値 成功すれば0以外，失敗すれば0。

DestroyWindow

定義 BOOL DestroyWindow (HWND Wnd) ;

ウィンドウまたはモードなしダイアログボックスとともに対応する子ウィンドウを破棄します。

引数 *Wnd*

ウィンドウハンドル。

戻り値 成功すれば0以外，失敗すれば0。

関連項目 ***CreateDialog, WM_DESTROY, WM_NCDESTROY***

DialogBox

定義 int DialogBox (HANDLE Instance, LPSTR TemplateName,
HWND WndParent, FARPROC DialogFunc) ;

TemplateName に基づきモードつきダイアログボックスを作成するとともに ***WM_INITDIALOG*** メッセージを送り，その後にダイアログを表示します。

引数 *Instance*

実行形式ファイルにダイアログボックステンプレートを含むモジュールインスタンス。

TemplateName

ダイアログボックステンプレート名（ヌルで終わる文字列)。

WndParent

オーナーウィンドウ。

DialogFunc

ダイアログコールバック関数の手続きインスタンスアドレス。

戻り値 ***EndDialog*** の ***Result*** 引数。ダイアログが作成されない場合は - 1。

D Func

DialogBoxIndirect

定義 int DialogBoxIndirect (HANDLE Instance, HANDLE DialogTemplate, HWND WndParent, FARPROC DialogFunc) ;

DialogTemplate に基づきモードつきダイアログボックスを作成するとともに ***WM_INITDIALOG*** メッセージを送り，その後にダイアログを表示します。

引数 *Instance*

実行形式ファイルにダイアログボックステンプレートを含むモジュールインスタンス。

DialogTemplate

DLGTEMPLATE 構造体を含むグローバルメモリブロック。

WndParent

オーナーウィンドウ。

DialogFunc

ダイアログコールバック関数の手続きインスタンスアドレス。

戻り値 ***EndDialog*** の ***Result*** 引数。ダイアログが作成されない場合は - 1。

DialogBoxIndirectParam

定義 int DialogBoxIndirectParam (HANDLE Instance, HANDLE DialogTemplate,
HWND WndParent, FARPROC DialogFunc,
DWORD InitParam) ;

DialogTemplate に基づきモードつきダイアログボックスを作成するとともに ***WM_INITDIALOG*** メッセージを送り，その後にダイアログを表示します。さらに初期化引数をコールバック関数に送ります。

引数 *Instance*

実行形式ファイルにダイアログボックステンプレートを含むモジュールインスタンス。

DialogTemplate

DLGTEMPLATE 構造体を含むグローバルメモリブロック。

WndParent

オーナーウィンドウ。

DialogFunc

ダイアログ関数の手続きインスタンスアドレス。

InitParam

WM_INITDIALOG メッセージの *lParam* として渡される引数。

戻り値 ***EndDialog*** の ***Result*** 引数。ダイアログが作成されない場合は－1。

DialogBoxParam

定義 int DialogBoxParam (HANDLE Instance, LPSTR TemplateName,
HWND Parent, FARPROC DialogFunc, DWORD InitParam) ;

TemplateName に基づきモードつきダイアログボックスを作成するとともに ***WM_INITDIALOG*** を送り，その後にダイアログを表示します。さらに初期

化引数をコールバック関数に送ります。

引数 *Instance*

実行形式ファイルにダイアログボックステンプレートを含むモジュールインスタンス。

TemplateName

ダイアログボックステンプレート名（ヌルで終わる文字列）。

Parent

オーナーウィンドウ。

DialogFunc

ダイアログ関数の手続きインスタンスアドレス。

InitParam

WM_INITDIALOG メッセージの ***lParam*** として渡される引数。

戻り値 ***EndDialog*** の ***Result*** 引数。ダイアログが作成されない場合は－1。

DispatchMessage

定義 DWORD DispatchMessage (LPMSG Msg) ;

Msg のメッセージをウィンドウ関数に渡します。

引数 *Msg*

MSG 構造体。

戻り値 ウィンドウ関数から返される値。ただし通常は無視。

DlgDirList

定義 int DlgDirList (HWND Dlg, LPSTR PathSpec, int IDListBox,
int IDStaticPath, WORD FileType) ;

PathSpec で指定したパス名と一致するファイルまたはディレクトリ一覧を *IDListBox* に格納します。

引数 *Dlg*

IDListBox を含むダイアログボックス。

PathSpec

パス名文字列（ヌルで終わる文字列)。

IDListBox

リストボックスコントロール ID。

IDStaticPath

カレントドライブとカレントディレクトリを表示するスタティックテキストコントロール ID。

FileType

0x0000 (読み込み/書き込み) , 0x0001 (読み込み専用) , 0x0002 (非表示) , 0x0004 (システム), 0x0010 (サブディレクトリ), 0x0020 (アーカイブ), 0x2000 (***LB_DIR***), 0x4000 (ドライブ), 0x8000 (排他ビット)。

戻り値 リストができればゼロ以外, パス検索が無効に終われば 0。

関連項目 ***LB_RESETCONTENT, LB_DIR***

DlgDirListComboBox

定義 int DlgDirListComboBox (HWND Dlg, LPSTR PathSpec, int IDComboBox, int IDStaticPath, WORD FileType) ;

PathSpec で指定したパス名と一致するファイルまたはディレクトリ一覧を *IDComboBox* に格納します。

引数 *Dlg*

IDComboBox を含むダイアログボックス。

PathSpec

パス名文字列（ヌルで終わる文字列）。

IDComboBox

コンボボックスコントロール ID。

IDStaticPath

カレントドライブとカレントディレクトリを表示するスタティックテキストコントロール ID。

FileType

0x0000 (読み込み/書き込み) , 0x0001 (読み込み専用) , 0x0002 (非表示) , 0x0004 (システム), 0x0010 (サブディレクトリ), 0x0020 (アーカイブ), 0x2000 (***LB_DIR***), 0x4000 (ドライブ), 0x8000 (排他ビット)。

戻り値 リストができればゼロ以外，パス検索が無効に終われば 0。

関連項目 ***CB_RESETCONTENT, CB_DIR***

DlgDirSelect

定義 BOOL DlgDirSelect (HWND Dlg, LPSTR Str, int IDListBox) ;

リストボックスの現選択項目を取得して，*Str* に代入します。

引数 *Dlg*

IDListBox を含むダイアログボックス。

Str

パス名バッファ。

IDListBox

リストボックスコントロール ID。

戻り値 現在の選択項目がディレクトリの場合はゼロ以外，そうでなければ 0。

関連項目 ***DlgDirList, LB_GETCURSEL, LB_GETTEXT***

DlgDirSelectComboBox

定義 BOOL DlgDirSelectComboBox (HWND Dlg, LPSTR Str, int IDComboBox) ;

単一コンボボックス ***CBS_SIMPLE*** からのみ現選択項目を取得して，*Str* に代入します。

引数 *Dlg*

IDComboBox を含むダイアログボックス。

Str

パス名バッファ。

IDComboBox

コンボボックスコントロール ID。

戻り値 現在の選択項目がディレクトリの場合は 0 以外，そうでなければ 0。

関連項目 ***DlgDirListComboBox, CB_GETCURSEL, CB_GETTEXT***

DOS3Call

定義 DOS3Call ;

MS-DOS のファンクションリクエスト，INT 21 h を呼び出します。INT 21 h 呼び出しのレジスタを設定しなければならないので，***DOS 3 Call*** の呼び出しはアセンブリ言語ルーチンからに限ってください。Windows では，***DOS 3 Call*** の方がソフトウェア割り込みを直接呼び出すよりも多少高速です。

DPtoLP

定義 BOOL DPtoLP (HDC DC, LPPOINT Points, int Count) ;

デバイス点(Device Point)を論理座標(Logical Point)に変換します。

引数 *DC*

デバイスコンテキストハンドル。

Points

POINT 構造体の配列。

Count

Points の点の数。

戻り値 すべての点が変換されれば 0 以外，そうでなければ 0。

DrawFocusRect

定義　void DrawFocusRect (HDC DC, LPRECT Rect) ;

XOR によりフォーカススタイル矩形を描きます。

引数　*DC*

デバイスコンテキストハンドル。

Rect

描かれる ***RECT*** 構造体。

DrawIcon

定義　BOOL DrawIcon (HDC DC, int X, int Y, HICON Icon) ;

アイコンを描きます。

引数　*DC*

デバイスコンテキストハンドル。

X, Y

アイコンの左上隅。

Icon

描かれるアイコン。

戻り値　成功すれば 0 以外，失敗すれば 0。

DrawMenuBar

定義　void DrawMenuBar (HWND Wnd) ;

ウィンドウのメニューバーを再描画します。この関数を使用するのは，ウィンドウ作成後のメニューバーに変更がある場合です。

引数 *Wnd*
ウィンドウハンドル。

DrawText

定義 int DrawText (HDC DC, LPSTR Str, int Count, LPRECT Rect,
WORD Format) ;

書式化テキストを描きます。書式タイプの指定は *Format* 引数によります。***DT_NOCLIP*** による指定が特になければ，矩形をはみ出すテキストはクリップされます。

引数 *DC*
デバイスコンテキストハンドル。

Str
描かれる文字列。*Count* が -1 の場合，ヌルで終わる文字列にすることが条件。

Count
文字列のバイト数。

Rect
矩形の座標をもつ ***RECT*** 構造体。

Format
次の定数を使用（第７章の「**DT_テキスト描画書式フラグ**」を参照)。

DT_BOTTOM, DT_CALCRECT, DT_CENTER, DT_EXPANDTABS, DT_EXTERNALLEADING, DT_LEFT, DT_NOCLIP, DT_NOPREFIX, DT_RIGHT, DT_SINGLELINE, DT_TABSTOP, DT_TOP, DT_VCENTER, DT_WORDBREAK

戻り値 テキストの高さ。

Ellipse

定義　BOOL Ellipse (HDC DC, int X 1, int Y 1, int X 2, int Y 2) ;

矩形の中央に楕円を描きます。楕円の境界線はペンで描き，内部はブラシで塗りつぶします。

引数　*DC*

デバイスコンテキストハンドル。

X 1, Y 1

矩形の左上隅。

X 2, Y 2

矩形の左下隅。

戻り値　楕円が描ければ 0 以外，描けなければ 0。

EmptyClipboard

定義　BOOL EmptyClipboard () ;

クリップボードを空にして，クリップボード内のデータのハンドルを解放します。クリップボードをオープンしているウィンドウに所有権が移ります。

戻り値　クリップボードが空になれば 0 以外，エラーの場合は 0。

EnableHardwareInput

定義 BOOL EnableHardwareInput (BOOL EnableInput) ;

EnableInput の指定どおりに入力の保存または廃棄を行ない，マウスまたはキーボードの入力を有効または無効にします。

引数 *EnableInput*

入力を保存する場合は 0 以外，廃棄する場合は 0。

戻り値 入力があらかじめ有効の場合は 0 以外（デフォルト），無効の場合は 0。

EnableMenuItem

定義 BOOL EnableMenuItem (HMENU Menu, WORD IDEnableItem,
WORD Enable) ;

メニュー項目の使用許可，使用禁止，淡色表示を *Enable* の指定どおりに行ないます。

引数 *Menu*

メニューハンドル。

IDEnableItem

チェック対象のメニュー項目またはポップアップの ID または位置。

Enable

定数 ***MF_COMMAND*** または ***MF_BYPOSITION*** のいずれかと ***MF_DISABLED, MF_ENABLED, MF_GRAYED*** のいずれかの組み合わせ（第 7 章の「**MF_メニューフラグ**」を参照）。

戻り値 メニュー項目の旧状態。メニュー項目が存在しない場合は -1。

EnableWindow

定義　BOOL EnableWindow (HWND Wnd, BOOL Enable) ;

ウィンドウまたはコントロールへのマウス入力またはキーボード入力を許可または禁止します。

引数　*Wnd*

許可または禁止にするウィンドウ。

Enable

許可の場合は 0 以外，禁止の場合は 0。

戻り値　成功の場合は 0 以外，失敗の場合は 0。

EndDeferWindowPos

定義　void EndDeferWindowPos (HANDLE WinPosInfo) ;

1 回の画面リフレッシュサイクルで，ウィンドウのサイズと位置を同時に変更します。

引数　*WinPosInfo*

ウィンドウの更新情報を含む複数のウィンドウデータの構造体。

関連項目　***BeginDeferWindowPos, DeferWindowPos***

EndDialog

定義 void EndDialog (HWND Dlg, int Result) ;

モードつきダイアログボックスを終了します。*Result* の指定値はダイアログボックスを作成する ***DialogBox*** 関数に返されます。

引数 *Dlg*

破棄されるダイアログ。

Result

戻り値。

EndPaint

定義 void EndPaint (HWND Wnd, LPPAINTSTRUCT Paint) ;

Wnd の描画終了を示します。

引数 *Wnd*

再描画ウィンドウ。

Paint

BeginPaint 関数から取得される ***PAINTSTRUCT***。

EnumChildWindows

定義　BOOL EnumChildWindows (HWND WndParent, FARPROC EnumFunc, DWORD lParam) ;

オーナーウィンドウの子ウィンドウを列挙します。コールバック関数に子ハンドルと *lParam* が渡されます。コールバックが 0 を返すか最後の子ウィンドウに到ると，列挙は終了します。

引数　*WndParent*

列挙対象となる子ウィンドウのオーナーウィンドウ。

EnumFunc

コールバック関数の手続きインスタンスアドレス。

lParam

コールバック関数に渡される値。

戻り値　子ウィンドウすべてが列挙されれば 0 以外，そうでなければ 0。

EnumClipboardFormats

定義　WORD EnumClipboardFormats (WORD Format) ;

使用可能なクリップボード書式を列挙します。

引数　*Format*

既知の書式名，ただしリストの最初の書式には 0。書式の指定を行なう *CF_*定数の詳細に関しては，第 7 章の「**CF_クリップボード書式**」を参照してください。

戻り値　クリップボード上の次の書式，ただし書式リストの最後またはクリップボードがクローズの場合は 0。

関連項目　***OpenClipboard***

EnumFonts

定義 int EnumFonts (HDC DC, LPSTR FaceName, FARPROC FontFunc,
LPSTR Data) ;

特定デバイスに使用可能な指定された名前のフォントを列挙します。***LOGFONT***, ***TEXTMETRIC***，フォント，***Data*** がコールバック関数に渡されます。コールバック関数が 0 を返すか，フォントが残らず列挙されれば，列挙は終了します。

引数 *DC*

デバイスコンテキストハンドル。

FaceName

ヌルで終わるフォント名。NULL を指定すると使用可能なフォントから 1 回に 1 つずつランダムに選択します。

FontFunc

コールバック関数の手続きインスタンスアドレス。

Data

コールバック関数に渡されるデータ。

戻り値 コールバックが返す最後の値。

EnumMetaFile

定義 BOOL EnumMetaFile (HDC DC, LOCALHANDLE MF,
FARPROC CallbackFunc, LPBYTE ClientData) ;

メタファイルの GDI 呼び出しを列挙します。コールバック関数に渡されるのは，*DC*，メタファイルのクラスハンドルテーブルのポインタ，メタファイルの構造体のポインタ，テーブル内に対応ハンドルを持つクラス数，*ClientData* です。コールバック関数が 0 を返すか，GDI 呼び出しがすべて列挙されれば，列挙は終了します。

引数 *DC*
メタファイルに対応するデバイスコンテキスト。

MF
メタファイルハンドル。

CallbackFunc
コールバック関数の手続きインスタンスアドレス。

ClientData
コールバック関数に渡されるデータ。

戻り値 メタファイルの GDI 呼び出しがすべて列挙されれば 0 以外，そうでなければ 0。

EnumObjects

定義 int EnumObjects (HDC DC, int ObjectType, FARPROC ObjectFunc,
LPSTR Data) ;

デバイスに使用できるオブジェクト型を列挙します。***LOGPEN*** または ***LOGBRUSH*** と *Data* がコールバック関数に渡されます。コールバック関数が 0 を返すか，クラスがすべて列挙されれば，列挙は終了します。

引数 *DC*
デバイスコンテキストハンドル。

ObjectType

OBJ_BRUSH または ***OBJ_PEN*** のいずれか（第7章の「**OBJ_オブジェクト型定数**」を参照）。

ObjectFunc

コールバック関数の手続きインスタンスアドレス。

Data

コールバック関数に渡されるデータ。

戻り値 コールバック関数から返される最後の値。

EnumProps

定義 int EnumProps (HWND Wnd, FARPROC EnumFunc) ;

ウィンドウのプロパティを列挙します。***Wnd***（プロパティに結びついた文字列）とデータハンドルがコールバック関数に渡されます。コールバック関数が0を返すか，プロパティすべてが列挙されれば，列挙は終了します。

引数 *Wnd*

ウィンドウハンドル。

EnumFunc

コールバック関数の手続きインスタンスアドレス。

戻り値 コールバック関数から返される最後の値。プロパティがなければ-1。

EnumTaskWindows

定義 BOOL EnumTaskWindows (HANDLE Task, EnumFunc FARPROC, DWORD lParam) ;

タスクのウィンドウをすべて列挙します。ウィンドウのハンドルと *lParam* がコールバック関数に渡されます。コールバックが 0 を返すか，ウィンドウすべてが列挙されれば，列挙は終了します。

引数 *Task*

タスクハンドル。

EnumFunc

コールバック関数の手続きインスタンスアドレス。

lParam

コールバック関数に渡される値。

戻り値 ウィンドウがすべて列挙されれば 0 以外，そうでなければ 0。

関連項目 ***GetCurrentTask***

EnumWindows

定義 BOOL EnumWindows (FARPROC EnumFunc, DWORD lParam) ;

画面の親ウィンドウをすべて列挙します。ウィンドウのハンドルと *lParam* がコールバック関数に渡されます。コールバック関数が 0 を返すか，ウィンドウがすべて列挙されれば，列挙は終了します。

引数 *EnumFunc*

コールバック関数の手続きインスタンスアドレス。

lParam

コールバック関数に渡される値。

戻り値 ウィンドウがすべて列挙されれば 0 以外，そうでなければ 0。

EqualRect

定義 BOOL EqualRect (LPRECT Rect 1, LPRECT Rect 2) ;

2つの矩形の左上隅と右下隅を比較して，両矩形が等しいかどうかを判定します。

引数 *Rect 1, Rect 2*
比較対象の2つの ***RECT*** 構造体。

戻り値 等しければ0以外，等しくなければ0。

EqualRgn

定義 BOOL EqualRgn (HRGN SrcRgn1, HRGN SrcRgn2) ;

2つの領域を比較し，等しいかどうかを判定します。

引数 *SrcRgn 1, SrcRgn 2*
比較対象の2つの領域。

戻り値 等しければ0以外，等しくなければ0。

Escape

定義 int Escape (HDC DC, int Escape, int Count, LPSTR InData, LPSTR OutData) ;

GDI がサポートしていないデバイス固有の機能へのアクセスを可能にします。

引数 *DC*

デバイスコンテキストハンドル。

Escape

Escape 関数。

Count

InData のデータバイト数。

InData

入力データ構造体。

OutData

Escape 出力データを受け取る構造体，出力データなしの場合は NULL。

戻り値 成功すれば正の値，エスケープがインプリメントされていなければ 0, エラーの場合は負の値。

エラーの原因として考えられるコードは，***SP_ERROR, SP_OUTOFDISK, SP_OUTOFMEMORY, SP_USERABORT***（第 7 章の「**SP_スプーラエラーコード**」を参照)。

EscapeCommFunction

定義 int EscapeCommFunction (int Cid, int Func) ;

Func の指定する通信デバイス拡張機能を実行します。

引数 *Cid*

通信デバイス。

Func

CLRDTR, CLRRTS, RESETDEV, SETDTR, SETRTS, SETXOFF, SETXON のいずれか（第7章の「**Escape comm（通信拡張）定数**」を参照）。

戻り値 成功すれば0, 無効機能コード指定の場合は負の値。

関連項目 ***OpenComm***

ExcludeClipRect

定義 int ExcludeClipRect (HDC DC, int X 1, int Y 1, int X 2, int Y 2) ;

既存領域から指定矩形を除いた，新クリッピング領域を作成します。

引数 *DC*

デバイスコンテキストハンドル。

X 1, Y 1

矩形の左上隅。

X 2, Y 2

矩形の右下隅。

戻り値 新領域のタイプは ***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION***（第7章の「**Region（領域）フラグ**」を参照）。

ExcludeUpdateRgn

定義 int ExcludeUpdateRgn (HDC DC, HWND Wnd) ;

ウィンドウのクリッピング領域から更新領域を除外して，非描画領域内での描画を防ぎます。

引数 *DC*

デバイスコンテキストハンドル。

Wnd

更新されるウィンドウ。

戻り値 ***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION*** の新領域タイプ（第7章の「**Region（領域）フラグ**」を参照)。

ExitWindows

定義 BOOL ExitWindows (DWORD Reserved, WORD ReturnCode) ;

Windows の標準シャットダウン処理を行ないます。すべてのアプリケーションが終了を受け入れて，Windows は終了します。DOS 割り込み 21 H ファンクション 4 CH が呼び出されます。

引数 *Reserved*

0 にセット。

ReturnCode

DOS に渡される値（AL)。

戻り値 アプリケーションの中で終了を受け入れないものがあれば 0。

関連項目 ***WM_QUERYENDSESSION, WM_ENDSESSION***

ExtDeviceMode

定義 int ExtDeviceMode (HWND Wnd, HANDLE Driver,
DEVMODE FAR * DevModeOutput,
LPSTR DeviceName, LPSTR Port,
DEVMODE FAR * DevModeInput,
LPSTR Profile, WORD Mode) ;

プリンタのデバイスドライバにデバイスの初期化情報を提供します。アプリケーションには，必ずDRIVINIT.H 含めてください。

引数 *Wnd*

ウィンドウハンドル（場合によっては，アプリケーションが呼び出したダイアログボックスの親ウィンドウ)。

Driver

デバイスドライバモジュールハンドル（***GetModuleHandle*** と ***LoadLibrary*** を参照)。

DevModeOutput

DEVMODE 構造体のポインタ。

DeviceName

ヌルで終わる文字列デバイス名のポインタ。

Port

デバイスに接続するポートを含む文字列のポインタ。

DevModeInput

ドライバ初期化情報を持つ ***DEVMODE*** 構造体のポインタ。

Profile

初期化ファイル名を含む文字列のポインタ。ヌルであれば，デフォルトは WIN.INI ファイル。

Mode

以下の操作の指定。

モード値	意味
0	**戻り値**を参照。
DM_COPY	ドライバの現在の設定を *DEVMODE* に書き込む。
DM_MODIFY	設定を変更する。
DM_PROMPT	ダイアログボックスからユーザ仕様を取得する。
DM_UPDATE	設定を環境と WIN.INI ファイルに書き込む

戻り値 *Mode* が 0 であれば，*DevModeInput* 構造体のバイト数。ダイアログボックスの場合は ***IDOK*** または ***IDCANCEL*** で，成功の場合に ***IDOK*** を返し，失敗の場合は 0 以下を返す。

関連項目 ***GetProcAddress***

ExtFloodFill

定義 BOOL ExtFloodFill (DC HDC, int X, int Y, COLORRFF Color,
DWORD FillType) ;

ラスタ表示面の一部を，*FillType* の指定方法で塗りつぶします。現在のブラシが使われます。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

塗りつぶし開始点。

Color

塗りつぶされる境界または領域の ***COLORREF*** (***DWORD***)。

FillType

FLOODFILLBORDER または ***FLOODFILLSURFACE*** のいずれか（第 7 章の「**Flood Fill（塗りつぶし）スタイルフラグ**」を参照）。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***FloodFill***

ExtTextOut

定義 BOOL ExtTextOut (HDC DC, int X, int Y, WORD Options, LPRECT Rect ;
LPSTR Str, int Count, LPINT Dx) ;

Rect 内に現在選択されているフォントの文字列を書き込みます。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

最初の文字セルの原点。

Options

場合によっては ***ETO_CLIPPED*** と ***ETO_OPAQUE*** の組み合わせ（第7章の「**ETO_ExtTextOut オプション**」を参照)。

ARect

RECT 構造体または NULL。

Str

書き込む文字列。

Count

文字列の文字数。

Dx

隣接セル間の間隔を指定する値の配列，デフォルト間隔の場合は 0。

戻り値 文字列が書ければ 0 以外，書けなければ 0。

関連項目 ***SetTextAlign***

FatalAppExit

定義　void FatalAppExit (WORD action, LPSTR MessageText) ;

アプリケーションを終了する最後の手段です。メッセージは表示しますが，メモリの解放とファイルのクローズに失敗するだけでなく，Windows 全体の失敗に陥る可能性があります。

引数　*Action*

予約されています。必ず 0 にセット。

MessageText

表示される 35 文字のポインタ。

FatalExit

定義　void FatalExit (int Code) ;

Code とスタックトレースを画面に表示し，コンピュータの AUX ポートにも出力します。ユーザには次の選択をするためのプロンプトが表示されます。これはデバッグを目的としたときにだけ使います。

引数　*Code*

表示エラーコード。

FillRect

定義　int FillRect (HDC DC, LPRECT Rect, HBRUSH Brush) ;

矩形の右の境界線と下の境界線までの領域を *Brush* で塗りつぶします。

引数　*DC*

デバイスコンテキストハンドル。

Rect

塗りつぶされる ***RECT*** 構造体。

Brush

塗りつぶしに使用するブラシ。

戻り値 ありません。

関連項目 ***CreateHatchBrush, CreatePatternBrush, CreateSolidBrush, GetStockObject***

FillRgn

定義 BOOL FillRgn (HDC DC, HRGN Rgn, HBRUSH Brush) ;

領域を ***Brush*** で塗りつぶします。

引数 *DC*

デバイスコンテキストハンドル。

Rgn

塗りつぶされる領域。

Brush

塗りつぶしに使用するブラシ。

戻り値 成功すれば 0 以外，失敗すれば 0。

FindAtom

定義 ATOM FindAtom (LPSTR Str) ;

Str に対応するアトムをアトムテーブルで検索します。

引数 *Str*

検索文字列（ヌルで終わる文字列)。

戻り値 *Str* に対応するアトム，テーブルに該当アトムがない場合は NULL。

FindResource

定義 HANDLE FindResource (HANDLE Instance, LPSTR Name,
LPSTR ResType) ;

リソースファイルのリソース位置を決定します。

引数 *Instance*

実行形式ファイルにリソースを含むモジュールインスタンス。

Name

リソース名（ヌルで終わる文字列または整数型 ID)。

ResType

リソース型を示す下記定数のいずれか（第 7 章の「**RT_リソース型**」を参照)。

RT_ACCELERATOR, RT_BITMAP, RT_CURSOR, RT_DIALOG, RT_FONT, RT_FONTDIR, RT_ICON, RT_MENU, RT_RCDATA, RT_STRING，ヌルで終わる文字列，整数型 ID。

戻り値 リソースハンドル，該当リソースがない場合は NULL。

FindWindow

定義 HWND FindWindow (LPSTR ClassName, LPSTR WindowName) ;

ClassName と *WindowName* が一致する最上位親ウィンドウを取得します。子ウィンドウの検索は行ないません。

引数 *ClassName*

ウィンドウクラス名（ヌルで終わる文字列），ただしウィンドウすべての場合はNULL。

WindowName

ウィンドウのテキストキャプション，ただしウィンドウすべての場合は NULL。

戻り値 ウィンドウハンドル，ただし該当ウィンドウがなければ NULL。

FlashWindow

定義 BOOL FlashWindow (HWND Wnd, BOOL Invert) ;

ウィンドウまたはアイコンを点滅させます。開いているウィンドウのアクティブ状態が反転されます。

引数 *Wnd*

点滅させるウィンドウ。

Invert

点滅させる場合は 0 以外，元の状態に戻す場合は 0。ただし後者はアイコンでは無視。

戻り値 呼び出し前のウィンドウがアクティブの場合は 0 以外，そうでなければ 0。

FloodFill

定義 BOOL FloodFill (HDC DC, int X, int Y, COLORRFF Color) ;

Color の境界までの表示領域を，現在のブラシで塗りつぶします。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

塗りつぶし開始位置。

Color

境界色の ***COLORREF***。

戻り値 成功すれば 0 以外，失敗すれば 0。

FlushComm

定義 int FlushComm (int Cid, int Queue) ;

通信デバイスの送受信キューをフラッシュします。

引数 *Cid*

フラッシュ対象の通信デバイス。

Queue

送信キューをフラッシュする場合は 0, 受信キューをフラッシュする場合は 1。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***OpenComm***

FrameRect

定義 void FrameRect (HDC DC, LPRECT Rect, HBRUSH Brush) ;

矩形の回りに 1 論理単位幅の境界線を描きます。

引数 *DC*

デバイスコンテキストハンドル。

Rect

矩形の左上隅と右上隅を定義する ***RECT*** 構造体。

Brush

境界線を描くブラシ。

関連項目 ***CreateHatchBrush, CreatePatternBrush, CreateSolidBrush***

FrameRgn

定義 BOOL FrameRgn (HDC DC, HRGN Rgn, HBRUSH Brush, int Width, int Height) ;

領域の境界線を描きます。

引数 *DC*

デバイスコンテキストハンドル。

Rgn

囲まれている領域。

Brush

境界線を描くブラシ。

Width

垂直ブラシストロークによる境界線の幅（論理単位)。

Height

水平ブラシストロークによる境界線の高さ（論理単位）。

戻り値　成功すれば 0 以外，失敗すれば 0。

FreeLibrary

定義　void FreeLibrary (HANDLE LibModule) ;

LibModule を無効にします。このモジュールを今後参照しない場合は，対応するメモリを解放します。

引数　*LibModule*

ロードされているライブラリモジュール。

FreeModule

定義　BOOL FreeModule (HANDLE Module) ;

Module を無効にします。このモジュールを今後参照しない場合は，対応するメモリを解放します。

引数　*Module*

ロードされているモジュールハンドル。

戻り値　ありません。

FreeProcInstance

定義　void FreeProcInstance (FARPROC Proc) ;

関数の手続きインスタンスアドレスを解放します。

引数 *Proc*

解放される関数の手続きインスタンスアドレス。

関連項目 ***MakeProcInstance***

FreeResource

定義 BOOL FreeResource (HANDLE ResData) ;

ResData を無効にします。このリソースを今後参照しない場合は，対応するメモリを解放します。

引数 *ResData*

リソースデータハンドル。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***LoadResource***

FreeSelector

定義 WORD FreeSelector (WORD Selector) ;

AllocSelector, AllocCStoDSAlias, AllocDStoCSAlias のいずれかで確保したセレクタを解放し，無効にします。

引数 *Selector*

解放するセレクタ。

戻り値 成功すれば 0, 失敗すれば *Selector*。

GetActiveWindow

定義 HWND GetActiveWindow () ;

現在入力フォーカスがあるウィンドウのハンドルを取得します。

戻り値 アクティブウィンドウハンドル。

関連項目 ***SetActiveWindow***

GetAspectRatioFilter

定義 DWORD GetAspectRatioFilter (HDC DC) ;

現在のアスペクト比フィルタが使用するアスペクト比を取得します。

引数 *DC*
指定アスペクト比を含むデバイスコンテキスト。

戻り値 アスペクト比（上位ワードでX座標，下位ワードでY座標)。

関連項目 ***SetMapperFlags***

GetAsyncKeyState

定義 int GetAsyncKeyState (int Key) ;

仮想キーの状態を判定します。

引数 *Key*
仮想キーコード。

戻り値 MSBがセットされていれば *Key* は押されている状態，LSBがセットされていれば *Key* が押されたのは直前の関数呼び出し後。

GetAtomHandle

定義　HANDLE GetAtomHandle (ATOM AnAtom) ;

指定アトムに対応する文字列を取得します。

引数　*AnAtom*

アトムハンドル。

戻り値　アトム文字列へのローカルメモリハンドル，ただし該当アトムがなければ0。

GetAtomName

定義　WORD GetAtomName (ATOM AnAtom, LPSTR Buffer, int Size) ;

対応するアトム文字列を *Buffer* にコピーします。

引数　*AnAtom*

アトムハンドル。

Buffer

アトム文字列を格納するバッファ。

Size

バッファサイズ（バイト単位)。

戻り値　*Buffer* にコピーされるバイト数，ただし無効アトム指定の場合は0。

GetBitmapBits

定義 LONG GetBitmapBits (HBITMAP Bitmap, LONG Count, LPSTR Bits) ;

ビットマップのビットを *Bits* にコピーします。

引数 *Bitmap*
ビットマップハンドル。

Count
コピーするバイト数。

Bits
水平走査ライン数が 16 ビットの倍数である構造体に対応するバイト配列。

戻り値 ビットマップの実際のバイト数，エラーの場合は 0。

GetBitmapDimension

定義 DWORD GetBitmapDimension (HBITMAP Bitmap) ;

ビットマップの高さと幅を取得します。

引数 *Bitmap*
ビットマップハンドル。

戻り値 上位ワードで高さ，下位ワードで幅（それぞれ 1/10 mm 単位）。
寸法が使用設定されていない場合，***SetBitmapDimension*** は 0。

関連項目 ***SetBitmapDimension***

GetBkColor

定義 DWORD GetBkColor (HDC DC) ;

現在のデバイスバックグラウンド色を取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 RGB カラー値。

GetBkMode

定義 int GetBkMode (HDC DC) ;

テキスト，ハッチブラシ，実線以外のペンスタイルに使用する現バックグラウンドモードを取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 ***OPAQUE*** または ***TRANSPARENT***（第 7 章の「**Background Mode（バックグラウンドモード）**」を参照）。

GetBrushOrg

定義 DWORD GetBrushOrg (HDC DC) ;

デバイスのカレントブラシの原点を取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 下位ワードで X 座標，上位ワードで Y 座標。

GetBValue

定義 BYTE GetBValue (DWORD RGBColor) ;

RGB カラー値から青の輝度値を取り出すマクロです。

引数 *RGBColor*
RGB カラー値。

戻り値 0 から 255 までの輝度の青の値。

GetCapture

定義 HWND GetCapture () ;

マウス入力の現在の送り先であるウィンドウを取得します。

戻り値 マウスキャプチャを持つウィンドウ，ただし該当ウィンドウがなければ 0。

関連項目 ***SetCapture***

GetCaretBlinkTime

定義 WORD GetCaretBlinkTime () ;

キャレットの点滅間隔であるキャレットブリンクを取得します。

戻り値 点滅速度（ミリ秒単位)。

GetCaretPos

定義 void GetCaretPos (LPPOINT Point) ;

現在のキャレット位置（クライアント座標系）を取得します。

引数 *Point*

格納する ***POINT*** 構造体。

GetCharWidth

定義 BOOL GetCharWidth (HDC DC, WORD FirstChar, WORD LastChar, LPINT Buffer) ;

指定の連続した文字グループの各文字幅を取得します。

引数 *DC*

デバイスコンテキストハンドル。

FirstChar

連続した文字グループの最初の文字。

LastChar

連続した文字グループの最後の文字。

Buffer

幅の値を格納する整数型配列。

戻り値 成功すれば 0 以外，失敗すれば 0。

GetClassInfo

定義 BOOL GetClassInfo (HANDLE Instance, LPSTR ClassInfo,
LPWNDCLASS WndClass) ;

指定クラスのクラス情報を取得します。***WNDCLASS*** フィールドの ***lpszClassName***, ***lpszMenuName***, ***hInstance*** は取得できません。

引数 *Instance*

クラスを作成したアプリケーションインスタンス，ただし定義済みの Windows クラスの場合は 0。

ClassName

クラス名（ヌルで終わる文字列）または ID。

WndClass

クラス情報を受け取る ***WNDCLASS***。

戻り値 成功すれば 0 以外，該当クラスがなければ 0。

GetClassLong

定義 LONG GetClassLong (HWND Wnd, int Index) ;

ウィンドウの ***WNDCLASS*** 構造体から，***Index*** 位置の long 値を取得します。正のバイトオフセット（0 以上）で，クラス情報の後ろに割り当てるバイトにアクセスします。

引数 *Wnd*

ウィンドウハンドル。

Index

バイトオフセット，または ***GCL_WNDPROC*** 定数（第 7 章の「**GCL_クラスフィールドオフセット**」を参照)。

戻り値 取得した値。

GetClassName

定義 int GetClassName (HWND Wnd, LPSTR ClassName, int MaxCount) ;

ウィンドウのクラス名を取得します。

引数 *Wnd*

ウィンドウハンドル。

ClassName

クラス名を格納するバッファ。

MaxCount

バッファサイズ。

戻り値 実際にコピーされた文字数，エラーの場合は 0。

GetClassWord

定義 WORD GetClassWord (HWND Wnd, int Index) ;

ウィンドウの ***WNDCLASS*** 構造体から，*Index* 位置の ***WORD*** 値を取得します。正のバイトオフセット（0 以上）で，クラス情報の後ろに割り当てるバイトにアクセスします。

引数 *Wnd*

ウィンドウハンドル。

Index

バイトオフセット，または次のいずれかの定数（第 7 章の「**GCW_クラスフィールドオフセット**」を参照)。

GCW_CBCLSEXTRA, GCW_CBWNDEXTRA, GCW_HBRBACKGROUND, GCW_HCURSOR, GCW_HICON, GCW_HMODULE, GCW_STYLE

戻り値 取得した値。

GetClientRect

定義 void GetClientRect (HWND Wnd, LPRECT Rect) ;

ウィンドウのクライアント座標系を取得します。

引数 *Wnd*
ウィンドウハンドル。

Rect
クライアント座標系を受け取る ***RECT*** 構造体。

GetClipboardData

定義 HANDLE GetClipboardData (WORD Format) ;

指定書式のクリップボードデータを取得します。取得されるメモリブロックはクリップボードによって制御されます。

引数 *Format*
クリップボードデータ書式で，***CF***_定数のいずれか（第7章の「**CF_クリップボード書式**」を参照）。

戻り値 クリップボードデータを含むメモリブロック，エラーの場合は NULL。

関連項目 ***SetClipboardData***

GetClipboardFormatName

定義 int GetClipboardFormatName (WORD Format, LPSTR FormatName,
MaxCount int) ;

クリップボードから登録された書式名を取得します。

引数 *Format*

クリップボード書式で，***CF***_定数の1つ（第7章の「**CF_クリップボード書式**」を参照)。

FormatName

格納するバッファ。

MaxCount

バッファサイズ。

戻り値 コピーされた文字列の実際の長さ，ただし要求された書式が無効の場合は0。

GetClipboardOwner

定義 HWND GetClipboardOwner () ;

クリップボードを現在所有するウィンドウを取得します。

戻り値 オーナーウィンドウ，ただしオーナーウィンドウがなければNULL。

GetClipboardViewer

定義 HWND GetClipboardViewer () ;

クリップボードビューアチェインの最初のウィンドウを取得します。

戻り値 現在クリップボードの表示を担当するウィンドウ，ただしビューアがなければNULL。

GetClipBox

定義 int GetClipBox (HDC DC, LPRECT Rect) ;

現在のクリッピング領域を囲む最小の境界矩形を取得します。

引数 *DC*

デバイスコンテキストハンドル。

Rect

格納する ***RECT*** 構造体。

戻り値 クリッピング領域のタイプで，***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION*** のいずれか（第 7 章の「**Region（領域）フラグ**」を参照）。

GetCodeHandle

定義 HANDLE GetCodeHandle (FARPROC Proc) ;

指定関数を含むコードセグメントを取得します。場合によっては，そのコードセグメントをロードします。

引数 *Proc*

関数の手続きインスタンスアドレス。

戻り値 関数を含むコードセグメント。

GetCodeInfo

定義 void GetCodeInfo (FARPROC Proc, LPVOID SegInfo) ;

Proc を含むコードセグメントに関する情報を取得します。

引数 *Proc*

関数のアドレス，またはモジュールハンドルとセグメント番号。

SegInfo

4 つの 32 ビット値で構成された配列。

GetCommError

定義 int GetCommError (int Cid, COMSTAT Stat) ;

デバイスの通信エラーをクリアします。

引数 *Cid*

通信デバイス。

Stat

デバイス状態の情報を受け取る ***COMSTAT***，または NULL。

戻り値 次のエラーコードのいずれか（第 7 章の「**CE_通信エラーフラグ**」を参照)。

CE_BREAK, CE_CTSTO, CE_DNS, CE_DSRTO, CE_FRAME, CE_IOE, CE_MODE, CE_OOP, CE_OVERRUN, CE_PTO, CE_RLSDTO, CE_RXOVER, CE_RXPARITY, CE_TXFULL

関連項目 ***OpenComm***

GetCommEventMask

定義 WORD GetCommEventMask (int Cid, int EvtMask) ;

デバイスの現イベントマスクを取得し，クリアします。

引数 *Cid*

通信デバイス。

EvtMask

使用可能なイベント。

戻り値 現在のイベントマスク値。

関連項目 ***OpenComm, SetCommEventMask***

GetCommState

定義 int GetCommState (Cid int, DCB DCB) ;

デバイスのデバイス制御ブロックを取得します。

引数 *Cid*

通信デバイス。

DCB

現在のデバイス制御ブロックを受け取る DCB。

戻り値 成功すれば 0, 失敗すれば負の値。

関連項目 ***OpenComm***

GetCurrentPDB

定義　WORD GetCurrentPDB () ;

現在の DOS プログラムデータベースである PDB を取得します。PDB は，プログラムセグメントプレフィクスとも呼ばれます。

戻り値　現在の PDB パラグラフアドレスまたはセレクタ。

GetCurrentPosition

定義　DWORD GetCurrentPosition (HDC DC) ;

現在の位置の論理座標を取得します。

引数　*DC*

デバイスコンテキストハンドル。

戻り値　下位ワードで X 座標，上位ワードで Y 座標。

GetCurrentTask

定義　HANDLE GetCurrentTask () ;

現在実行中のタスクのハンドルを取得します。

戻り値　成功すればタスクハンドル，失敗すれば NULL。

GetCurrentTime

定義 DWORD GetCurrentTime () ;

システムのリブート後の経過時間を取得します。

戻り値 現在の経過時間（ミリ秒単位）。

GetCursorPos

定義 void GetCursorPos (LPPOINT Point) ;

現在のカーソル位置の画面座標を取得します。

引数 *Point*

格納する ***POINT*** 構造体。

GetDC

定義 HDC GetDC (HWND Wnd) ;

クライアント領域で GDI 関数が描画を行なうためのディスプレイコンテキストを取得します。

引数 *Wnd*

ウィンドウハンドル。

戻り値 ディスプレイコンテキストハンドル，ただしエラーの場合は NULL。

関連項目 ***ReleaseDC***

GetDCOrg

定義 DWORD GetDCOrg (HDC DC) ;

デバイスの最終変換原点（画面座標系）を取得します。この原点は，デバイス座標系からクライアント座標系への変換に使用されるオフセットです。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 下位ワードでX座標，上位ワードでY座標。

GetDesktopWindow

定義 HWND GetDesktopWindow () ;

デスクトップウィンドウへのハンドルを取得します。

戻り値 Windowsのデスクトップウィンドウハンドル。

GetDeviceCaps

定義 int GetDeviceCaps (HDC DC, int Index) ;

ディスプレイ固有の情報を取得します。

引数 *DC*

デバイスコンテキストハンドル。

Index

返される項目（第7章の「**Device capabilities（デバイス機能）**」を参照）。

戻り値 要求項目に対する値。

GetDialogBaseUnits

定義 LONG GetDialogBaseUnits () ;

ダイアログの基本単位を取得します。基本幅は，システムの平均フォント幅を表わします。実際のダイアログ単位の幅と高さは，それぞれ返される幅の 4 分の 1 と高さの 8 分の 1 です。

戻り値 上位ワードで高さの基本単位，下位ワードで幅の基本単位(それぞれピクセル数)。

GetDIBits

定義 int GetDIBits (HDC DC, HANDLE Bitmap, WORD StartScan,
WORD NumScans, LPSTR Bits, LPBITMAPINFO BitInfo,
WORD Usage) ;

ビットマップをデバイス独立の書式で *Bits* にコピーします。

引数 *DC*

デバイスコンテキストハンドル。

Bitmap

ソースビットマップハンドル。

StartScan

最初の走査ライン。

NumScans

コピーする行数。

Bits

ビットマップを格納するバッファ，ただし *BitsInfo* に代入する場合は NULL。

BitInfo

ビットマップのカラー書式と寸法を含む ***BITMAPINFO*** 構造体。

Usage
カラーのソースを示す ***DIB_PAL_COLORS*** または ***DIB_RGB_COLORS*** のいずれかの定数（第7章の「**DIB_カラーテーブル識別子**」を参照)。

戻り値 コピーされた走査ライン数，エラーの場合は0。

GetDlgCtrlID

定義 int GetDlgCtrlID (HWND Wnd) ;

コントロールウィンドウの ID 値を取得します。

引数 *Wnd*
コントロールハンドル。

戻り値 コントロールの数値ハンドル，ただしエラーの場合は0。

GetDlgItem

定義 HWND GetDlgItem (HWND Dlg, int IDDlgItem) ;

指定ダイアログボックスに含まれるコントロールのハンドルを取得します。

引数 *Dlg*
コントロールを含むダイアログボックス。

IDDlgItem
コントロール ID。

戻り値 コントロールハンドル，ただし指定コントロールが存在しなければ NULL。

GetDlgItemInt

定義 WORD GetDlgItemInt (HWND Dlg, int IDDlgItem, BOOL Far *Translate, BOOL signed) ;

ダイアログボックスのコントロールテキストを，整数値に変換します。直前の空白は除去されます。

引数 *Dlg*

ダイアログボックスハンドル。

IDDlgItem

項目 ID。

Translate

返される論理値，ただし変換エラーの場合は 0。

Signed

符号つきの値として扱われる取得値。

戻り値 変換した値。

関連項目 ***WM_GETTEXT***

GetDlgItemText

定義 int GetDlgItemText (HWND Dlg, int IDDlgItem, LPSTR Str, MaxCount int) ;

コントロールのテキストを取得します。

引数 *Dlg*

ダイアログボックスハンドル。

IDDlgItem

項目 ID。

Str

テキストを格納するバッファ。

MaxCount

バッファサイズ。

戻り値 実際にコピーされた文字数。

関連項目 ***WM_GETTEXT***

GetDOSEnvironment

定義 LPSTR GetDOSEnvironment () ;

現在のタスクの DOS 環境文字列を取得します。

戻り値 タスクの環境文字列。

GetDoubleClickTime

定義 WORD GetDoubleClickTime () ;

ダブルクリックの，最初のクリックと次のクリック間の最大時間を取得します。

戻り値 現在のダブルクリック時間（ミリ秒単位)。

GetDriveType

定義 WORD GetDriveType (int Drive) ;

Drive が交換可能タイプか，固定タイプか，リモートタイプかを判定します。

引数 *Drive*

調べるドライブ（Aドライブであれば0，Bドライブであれば1など）。

戻り値 ***DRIVE_REMOVEABLE, DRIVE_FIXED, DRIVE_REMOTE*** のいずれか，判定不能の場合は0，該当ドライブが存在しなければ1（第7章の「**DRIVE_ドライブタイプ**」を参照）。

GetEnvironment

定義 int GetEnvironment (LPSTR PortName, LPSTR Environ,
WORD MaxCount) ;

システムポートと結びつけられたデバイスの，現在の環境を取得します。

引数 *PortName*

ポート名（ヌルで終わる文字列）。

Environ

環境を格納するバッファ（第1フィールドをデバイス名とする），ただし必要サイズを返す場合はNULL。

MaxCount

バッファサイズ。

戻り値 実際にコピーされたバイト数，ただし該当環境がなければ0。

GetFocus

定義 HWND GetFocus () ;

現在入力フォーカスがあるウィンドウを取得します。

戻り値 成功すればウィンドウハンドル，失敗すればNULL。

GetFreeSpace

定義 DWORD GetFreeSpace (WORD Flag) ;

グローバルヒープの空きメモリ量を取得します。

引数 *Flag*

バンクラインの下方を走査する場合は ***GMEM_NOT_BANKED*** 定数，バンクラインの上方を走査する場合は0, ただしEMS以外のシステムでは無視（第7章の「**GMEM_グローバルメモリフラグ**」を参照)。

戻り値 使用可能メモリ（バイト単位)。

関連項目 ***GlobalCompact***

GetGValue

定義 BYTE GetGValue (DWORD RGBColor) ;

RGBカラー値から緑の輝度値を抜き出すマクロです。

引数 *RGBColor*

RGBカラー値。

戻り値 0から255までの輝度の緑の輝度値。

GetInputState

定義　BOOL GetInputState () ;

システムキューにマウス，キーボード，タイマの各イベントが現在あるかどうかを判定します。

戻り値　入力が検出されれば 0 以外，検出されなければ 0。

GetInstanceData

定義　int GetInstanceData (HANDLE Instance, NPSTR Data, int Conut) ;

前回のインスタンスデータを *Data* にコピーします。

引数　*Instance*
前回のアプリケーションのインスタンスハンドル。

Data
格納するバッファ。

Count
バッファサイズ。

戻り値　実際にコピーされたバイト数。

GetKBCodePage

定義　int GetKBCodePage () ;

現在ロードされている OEM/ANSI テーブルを取得します。

戻り値　現在のコードページ。

米国は437, 各国共通は850, ポルトガルは860, アイスランドは861, フランス系カナダは863, ノルウェー/デンマークは865。

GetKeyboardState

定義 void GetKeyboardState (BYTE KeyState) ;

KeyState に仮想キーボードのキー状態をコピーします。バイトの上位ビットが1の場合は，キーが押されている状態を示します。下位ビットが1の場合は，システム起動後に奇数回キーが押されていることを示します。

引数 *KeyState*
256バイトの文字配列。

GetKeyboardType

定義 int GetKeyboardType (int TypeFlag) ;

システムキーボードのタイプを取得します。

引数 *TypeFlag*
キーボードタイプは0, キーボードサブタイプは1, ファンクションキー（FK）数は2。

戻り値 キーボードタイプは次の値。

10 FKのPC/XTは1, 12 FKのOlivetti M 24は2, 10 FKのATは3, 12 FKのEnhancedは4, 10 FKのNokia 1050は5, 24 FKのNokia 9140は6, 日本語対応キーボードは7。

キーボードサブタイプは次の値。

PC-9800シリーズ一般のキーボードは0D01，PC 9801 VX/UX，PC-98 XL/XL 2用キーボード（Nモード）は，0D02，PC 98-XL/XL 2用キーボード（Hモード）は0D03，PC-9800シリーズラップトップ用キーボードは0D04。

GetKeyNameText

定義 int GetKeyNameText (DWORD lParam, LPSTR Buffer, int Size) ;

キー名が2文字以上の文字列を取得します。

引数 *lParam*

WM_KEYDOWN メッセージの long 引数。

Buffer

格納するバッファ。

Size

バッファサイズ。

戻り値 実際にコピーされたバイト数。

GetKeyState

定義 int GetKeyState (int VirtKey) ;

仮想キーが押されているかいないか，ロックされているかのいずれかの状態を取得します。

引数 *VirtKey*

仮想キー。

戻り値 上位ビットが1の場合のキーは押された状態，下位ビットが1の場合はロックされた状態。

GetLastActivePopup

定義　HWND GetLastActivePopup (HWND WndOwner) ;

直前のアクティブポップアップを取得します。

引数　*WndOwner*

ポップアップのオーナーウィンドウ。

戻り値　ポップアップウィンドウハンドルまたは *WndOwner*。

GetMapMode

定義　int GetMapMode (HDC DC) ;

現在のマッピングモードを取得します。

引数　*DC*

デバイスコンテキストハンドル。

戻り値　マッピングモードの *MM*_定数（第7章の「**MM_マッピングモード**」を参照)。

関連項目　***SetMapMode***

GetMenu

定義　HMENU GetMenu (HWND Wnd) ;

ウィンドウのメニューハンドルを取得します。

引数　*Wnd*

メニューのオーナーウィンドウ。

戻り値　メニューハンドル，メニューがなければ NULL，*Wnd* が子ウィンドウの場合は未定義。

GetMenuCheckMarkDimensions

定義 DWORD GetMenuCheckMarkDimensions () ;

メニュー項目の横に表示されるデフォルトチェックマークのビットマップ寸法を取得します。

戻り値 上位ワードで高さ，下位ワードで幅（それぞれピクセル単位)。

関連項目 ***SetMenuItemBitmaps***

GetMenuItemCount

定義 WORD GetMenuItemCount (HMENU Menu) ;

指定メニューの最上位メニュー数とメニュー項目数を取得します。

引数 *Menu*

メニューハンドル。

戻り値 成功すればメニュー項目数，失敗すれば - 1。

GetMenuItemID

定義 WORD GetMenuItemID (HMENU Menu, int Pos) ;

メニューの指定位置にあるメニュー項目 ID を取得します。

引数 *Menu*

ポップアップメニューハンドル。

Pos

メニュー項目の位置（0 からはじまる)。

戻り値 成功すれば項目 ID，項目がさらにポップアップを呼び出す場合は 0, エラーの場合は - 1。

GetMenuState

定義 WORD GetMenuState (HMENU Menu, WORD Id, WORD Flags) ;

指定メニュー項目の状態情報を取得します。

引数 *Menu*

メニューまたはポップアップメニューハンドル。

ID

メニュー項目 ID。

Flags

MF_BYPOSITION または ***MF_BYCOMMAND*** のいずれかのメニュー定数（第 7 章の「**MF_メニューフラグ**」を参照)。

戻り値 次の値のフラグマスク。

MF_CHECKED, MF_DISABLED, MF_ENABLED, MF_MENUBARBREAK, MF_MENUBREAK, MF_SEPARATOR, MF_UNCHECKED。ポップアップの場合は上位バイトで項目数，ID が無効の場合は - 1。

GetMenuString

定義 int GetMenuString (HMENU Menu,WORD IDItem, LPSTR Str,
int MaxCount, WORD Flag) ;

メニュー項目のラベルを *Str* にコピーします。コピーされたラベルの終わりはヌルになります。

引数 *Menu*

メニューハンドル。

IDItem

メニュー項目 ID。

Str

格納するバッファ。

MaxCount

バッファサイズ。

Flag

MF_BYPOSITION または ***MF_BYCOMMAND*** のいずれかのメニュー定数(第7章の「**MF_メニューフラグ**」を参照)。

戻り値 実際にコピーされたバイト数。

GetMessage

定義 BOOL GetMessage (LPMSG Msg, HWND Wnd, WORD MsgFilterMin,
WORD MsgFilterMax) ;

アプリケーションキューのメッセージから，フィルタ範囲内のメッセージを取得します。該当メッセージがない場合，または後続メッセージが ***WM_PAINT*** か ***WM_TIMER*** の場合には，他のアプリケーションに制御権を譲ります。

引数 *Msg*
格納する ***MSG*** 構造体。

Wnd
メッセージ送り先ウィンドウ，ただしアプリケーションのウィンドウすべてに送る場合は 0。

MsgFilterMin
フィルタなしの場合は 0, キーボードのみの場合は ***WM_KEYFIRST***，マウスのみの場合は ***WM_MOUSEFIRST***。

MsgFilterMax
フィルタなしの場合は 0, キーボードのみの場合は ***WM_KEYLAST***，マウスのみの場合は ***WM_MOUSELAST***。

戻り値 ***WM_QUIT*** メッセージ以外は 0 以外，***WM_QUIT*** は 0。

GetMessagePos

定義 DWORD GetMessagePos () ;

GetMessage からの最終メッセージを取得した時点のカーソル位置を取得します。

戻り値 下位ワードで X 座標，上位ワードで Y 座標。

GetMessageTime

定義 DWORD GetMessageTime () ;

GetMessage により取得する最終メッセージまでの，システムリブート後の経過時間を取得します。

戻り値 メッセージ時間（ミリ秒単位)。

GetMetaFile

定義 HANDLE GetMetaFile (LPSTR FileName) ;

指名メタファイルのハンドルを作成します。

引数 *FileName*

メタファイルの DOS ファイル名（ヌルで終わる文字列)。

戻り値 成功すればメタファイルハンドル，失敗すれば 0。

GetMetaFileBits

定義 HANDLE GetMetaFileBits (HANDLE MF) ;

メタファイルをビット列として含む，読み込み専用グローバルメモリブロックを取得します。メタファイルのサイズの判定とセーブに使用します。

引数 *MF*

メモリメタファイルハンドル。呼び出し後は無効。

戻り値 成功すればグローバルメモリブロック，失敗すれば 0。

GetModuleFileName

定義 int GetModuleFileName (HANDLE Module, LPSTR FileName, int Size) ;

指定モジュールの実行形式ファイルのフルパス名（ヌルで終わる文字列）を取得します。

引数 *Module*

モジュールハンドル。

FileName

格納するバッファ。

Size

バッファサイズ。

戻り値 実際にコピーされたバイト数。

GetModuleHandle

定義 HANDLE GetModuleHandle (LPSTR ModuleName) ;

モジュールのハンドルを取得します。

引数 *ModuleName*

モジュール名（ヌルで終わる文字列）。

戻り値 成功すればモジュールハンドル，失敗すれば0。

GetModuleUsage

定義 int GetModuleUsage (HANDLE Module) ;

モジュールの参照カウントを取得します。

引数 *Module*

モジュールハンドル。

戻り値 参照カウント。

GetNearestColor

定義　DWORD GetNearestColor (HDC DC, COLORREF Color) ;

デバイスがサポートできる *Color* に最も近い論理カラーを取得します。

引数　*DC*

デバイスコンテキストハンドル。

Color

照合される ***COLORREF***。

戻り値　RGB ソリッドカラー。

GetNearestPaletteIndex

定義　WORD GetNearestPaletteIndex(HPALETTE Palette, COLORREF Color) ;

論理パレットの中で *Color* に最も近い色を取得します。

引数　*Palette*

論理パレットハンドル。

Color

照合される ***COLORREF***。

戻り値　論理パレットのエントリインデックス。

GetNextDlgGroupItem

定義　HWND GetNextDlgGroupItem (HWND Dlg, HWND Ctrl, BOOL Previous) ;

次の（または前の）***WS_GROUP*** スタイルコントロールを *Ctrl* から取得します。取得のための検索は，周期的に繰り返されます。

引数 *Dlg*

ダイアログボックスハンドル。

Ctrl

検索開始コントロールのハンドル。

Previous

前のコントロールを検索する場合は 0, 次のコントロールを検索する場合は 0 以外。

戻り値 コントロールハンドル。

GetNextDlgTabItem

定義 HWND GetNextDlgTabItem (HWND Dlg, HWND Ctrl, BOOL Previous) ;

次の（または前の） ***WS_TABSTOP*** スタイルコントロールを *Ctrl* から取得します。取得のための検索は，周期的に繰り返されます。

引数 *Dlg*

ダイアログボックスハンドル。

Ctrl

検索開始コントロールのハンドル。

Previous

前のコントロールを検索する場合は 0, 次のコントロールを検索する場合は 0 以外。

戻り値 コントロールハンドル。

GetNextWindow

定義 HWND GetNextWindow (HWND Wnd, WORD Flag) ;

次の（または前の）ウィンドウを ***Wnd*** から取得します。最上位ウィンドウの場合は次の最上位ウィンドウを，子ウィンドウの場合は次の子ウィンドウを検索します。

引数 *Wnd*
ウィンドウハンドル。

Flag
GW_HWNDNEXT または ***GW_HWNDPREV*** のいずれかの定数（第 7 章の「**GW_ウィンドウ取得定数**」を参照）。

戻り値 ウィンドウハンドル。

GetNumTasks

定義 WORD GetNumTasks () ;

システムで実行中のタスク数を取得します。

戻り値 実行中のタスク数。

GetObject

定義 int GetObject (HANDLE hObject, int Count, LPSTR Object) ;

論理オブジェクトを定義するデータをバッファに入れます。論理パレットのエントリ数のみが返されます。

引数 *hObject*
オブジェクトハンドル。

Count

バッファサイズ（*ObjectPtr* にコピーするバイト数）。

ObjectPtr

格納するバッファ ***LOGPEN, LOGBRUSH, LOGFONT, BITMAP*** または整数型のいずれか。

戻り値 実際にコピーされたバイト数，ただしエラーの場合は 0。

関連項目 ***GetBitmapBits, GetPaletteEntries***

GetPaletteEntries

定義

```
WORD GetPaletteEntries (HPALETTE Palette, WORD StartIndex,
                        WORD NumEntries,
                        LPPALETTEENTRY PaletteEntries) ;
```

パレットエントリの指定範囲を取得して，*PaletteEntries* にコピーします。

引数 *Palette*

論理パレットハンドル。

StartIndex

最初のエントリ。

NumEntries

エントリ数。

PaletteEntries

パレットエントリを格納する ***PALETTEENTRY*** 構造体の配列。

戻り値 実際に取得されたエントリ数，ただしエラーの場合は 0。

GetParent

定義 HWND GetParent (HWND Wnd) ;

親ウィンドウのハンドルを取得します。

引数 *Wnd*

ウィンドウハンドル。

戻り値 親ウィンドウハンドル，ただし親ウィンドウがない場合は 0。

GetPixel

定義 DWORD GetPixel (HDC DC, int X, int Y) ;

指定座標位置の RGB カラーを取得します。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

検査対象の座標位置。

戻り値 RGB カラー値，ただしクリッピング領域に座標位置がない場合は - 1。

GetPolyFillMode

定義 int GetPolyFillMode (HDC DC) ;

現在の多角形塗りつぶしモードを取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 ***ALTERNATE*** または ***WINDING*** の多角形塗りつぶしモード（第7章の「**PolyFill モード**」を参照）。

GetPriorityClipboardFormat

定義 int GetPriorityClipboardFormat (WORD FAR *PriorityList, int Count) ;

PriorityList の最初のクリップボード書式を取得します。

引数 *PriorityList*

CF_定数のクリップボード書式が優先順に並ぶ整数型配列（第7章の「**CF_クリップボード書式**」を参照）。

Count

PriorityList のサイズ。

戻り値 リストの最優先書式，ただし該当書式がなければ -1。クリップボードになれければ 0。

GetPrivateProfileInt

定義 WORD GetPrivateProfileInt (LPSTR ApplicationName, LPSTR KeyName, int Default, LPSTR FileName) ;

指定の初期化ファイルから整数型キー値を取得します。

引数 *ApplicationName*

FileName のアプリケーションヘッダ名。

KeyName

FileName のキー名。

Default

該当 *KeyName* がない場合のデフォルト値。

FileName

Windows ディレクトリの初期化ファイル名。

戻り値　キー値，ただし値が負または整数型でない場合は 0。

GetPrivateProfileString

定義　int GetPrivateProfileString (LPSTR ApplicationName, LPSTR KeyName,
LPSTR Default, LPSTR ReturnedString,
int Size, LPSTR FileName) ;

指定の初期化ファイルから文字列キー値を取得します。

引数　*ApplicationName*

FileName のアプリケーションヘッダ名。

KeyName

FileName のキー名。キー名リストを取得する場合は NULL。

Default

該当 *KeyName* がない場合のデフォルト値。

ReturnedString

格納するバッファ。

Size

バッファサイズ。

FileName

Windows ディレクトリの初期化ファイル名。

戻り値　実際にコピーされた文字数。

GetProcAddress

定義 FARPROC GetProcAddress (HANDLE Module, LPSTR ProcName) ;

エクスポートされるライブラリ関数のアドレスを取得します。

引数 *Module*

ライブラリモジュール。

ProcName

関数名（ヌルで終わる文字列），または序数名。

戻り値 成功すれば関数のエントリポイント，失敗すれば0。

GetProfileInt

定義 WORD GetProfileInt (LPSTR AppName, LPSTR KeyName, int Default) ;

WIN.INI ファイルから整数型キー値を取得します。

引数 *AppName*

アプリケーションヘッダ名。

KeyName

検索するキー名。

Default

該当 *KeyName* がない場合のデフォルト値。

戻り値 キー値，ただし値が負であるかまたは整数型でない場合は0。

GetProfileString

定義　int GetProfileString (LPSTR AppName, LPSTR KeyName, LPSTR Default, LPSTR ReturnedString, int Size) ;

WIN.INI ファイルから文字列キー値を取得します。

引数　*AppName*

アプリケーションヘッダ名。

KeyName

取得キー名。*AppName* に対応するキー名すべてを取得する場合は NULL。

Default

該当 *KeyName* がない場合のデフォルト値。

ReturnedString

格納するバッファ。

Size

バッファサイズ。

戻り値　実際にコピーされた文字数。

GetProp

定義　HANDLE GetProp (HWND Wnd, LPSTR Str) ;

ウィンドウのプロパティリストから，対応するデータハンドルを取得します。

引数　*Wnd*

ウィンドウハンドル。

Str

ヌルで終わる文字列またはアトム。

戻り値　プロパティリストに *Str* があればデータハンドル，なければ 0。

関連項目 *AddAtom*

GetRgnBox

定義 int GetRgnBox (HRGN Rgn, LPRECT Rect) ;

領域の境界矩形を取得します。

引数 *Rgn*

領域ハンドル。

Rect

格納する ***RECT***。

戻り値 ***COMPLEXREGION, NULLREGION, SIMPLEREGION*** のいずれかの領域タイプ，無効領域の場合は 0（第 7 章の「**Region（領域）フラグ**」を参照)。

GetROP 2

定義 int GetROP 2 (HDC DC) ;

現在の描画モードを取得します。

引数 *DC*

ラスタデバイスコンテキスト。

戻り値 ***R 2***_定数のいずれかの描画モード（第 7 章の「**R 2_バイナリラスタ操作**」を参照)。

関連項目 ***SetROP 2***

GetRValue

定義 BYTE GetRValue (DWORD RGBColor) ;

RGB カラー値から赤の輝度値を抜き出すマクロです。

引数 *RGBColor*

RGB カラー値。

戻り値 0 から 255 までの赤の輝度値。

GetScrollPos

定義 int GetScrollPos (HWND Wnd, int Bar) ;

現在のスクロール範囲に対するスクロールボックスの位置を取得します。

引数 *Wnd*

スクロールバーを含むウィンドウ。

Bar

SB_CTL, SB_HORZ, SB_VERT のいずれかの定数（第 7 章の「**SB_スクロールバー定数**」を参照）。

戻り値 スクロールボックスの現在位置。

GetScrollRange

定義 void GetScrollRange (HWND Wnd, int Bar, LPINT MinPos,
LPINT MaxPos) ;

スクロールバー位置の最小値と最大値を取得します。

引数 *Wnd*

スクロールバーを含むウィンドウ。

Bar

SB_CTL, SB_HORZ, SB_VERT のいずれかの定数（第7章の「**SB_スクロールバー定数**」を参照)。

MinPos

位置の最小値を格納する整数型。

MaxPos

位置の最大値を格納する整数型。

GetStockObject

定義 HANDLE GetStockObject (int Index) ;

定義済みのストックペン，ブラシ，フォントのハンドルを取得します。

引数 *Index*

次の定数のいずれか（第7章の「**Stock logical object（ストック論理オブジェクト)**」を参照)。

BLACK_BRUSH, DKGRAY_BRUSH, GRAY_BRUSH, HOLLOW_BRUSH, LTGRAY_BRUSH, NULL_BRUSH, WHITE_BRUSH, BLACK_PEN, NULL_PEN, WHITE_PEN, ANSI_FIXED_FONT, ANSI_VAR_FONT, DEVICE_DEFAULT_FONT, OEM_FIXED_FONT, SYSTEM_FONT, SYSTEM_FIXED_FONT, DEFAULT_PALETTE

戻り値 成功すれば論理オブジェクトハンドル，失敗すれば0。

GetStretchBitMode

定義 int GetStretchBitMode (HDC DC) ;

現在のストレッチングモードを取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 ***WHITEONBLACK, BLACKONWHITE, COLORONCOLOR*** のいずれかの定数（第 7 章の「**StretchBlt モード**」を参照)。

関連項目 ***SetStretchBltMode***

GetSubMenu

定義 HMENU GetSubMenu (HMENU Menu, int Pos) ;

ポップアップメニューのハンドルを取得します。

引数 *Menu*

メニューハンドル。

Pos

Menu 内のポップアップメニュー位置。

戻り値 ポップアップメニューハンドル，ただし *Pos* にポップアップがない場合は 0。

GetSysColor

定義 DWORD GetSysColor (int Index) ;

表示要素の現在の色を取得します。

引数 *Index*

表示要素。

戻り値 RGB カラー値。

関連項目 ***SetSysColor***

GetSysModalWindow

定義 HWND GetSysModalWindow () ;

システムモードつきウィンドウのハンドルを取得します。

戻り値 システムモードつきウィンドウがある場合はそのハンドル，ない場合は 0。

GetSystemDirectory

定義 void GetSystemDirectory (LPSTR Buffer, WORD Size) ;

Windows システムサブディレクトリのパス名を取得します。

引数 *Buffer*

格納するバッファ。

Size

最低 144 文字を格納できるバッファサイズ。

GetSystemMenu

定義 HMENU GetSystemMenu (HWND Wnd, BOOL Revert) ;

システムメニューを取得してコピーおよび変更を行ないます。

引数 *Wnd*

ウィンドウハンドル。

Revert

システムメニューのコピーへのハンドルを返す場合は 0, 修正前の元のシステムメニューへのハンドルを返す場合は 0 以外。

戻り値 システムメニューハンドル，ただし *Revert* が 0 以外でシステムメニューに変更がない場合は 0。

関連項目 ***AppendMenu, InsertMenu, ModifyMenu***

GetSystemMetrics

定義 int GetSystemMetrics (int Index) ;

各種表示要素のピクセル幅と高さ，マウスステータス，Windows のデバッグバージョンなどのシステムメトリックスを取得します。

引数 *Index*

*SM*_定数のいずれか（第 7 章の「**SM_システムメトリックスコード**」を参照)。

戻り値 要求したシステムメトリック値。

GetSystemPaletteEntries

定義 WORD GetSystemPaletteEntries (HDC DC, WORD StartIndex,
WORD NumEntries,
LPPALETTEENTRY PaletteEntries) ;

システムパレットから一定範囲のパレットエントリ数を取得します。

引数 *DC*
デバイスコンテキストハンドル。

StartIndex
最初の取得エントリ。

NumEntries
取得エントリ数。

PaletteEntries
格納する ***PALETTEENTRY*** 構造体の配列。

戻り値 実際に取得されたエントリ数，ただしエラーの場合は 0。

GetSystemPaletteUse

定義 WORD GetSystemPaletteUse (HDC DC) ;

アプリケーションからのシステムパレットへのアクセスが完全かどうかを判定します。

引数 *DC*
デバイスコンテキストハンドル。

戻り値 ***SYSPAL_NOSTATIC*** または ***SYSPAL_STATIC*** のいずれかの定数（第7章の「**SYSPAL_システムパレットフラグ**」を参照)。

関連項目 ***SetSystemPaletteUse***

GetTabbedTextExtent

定義 DWORD GetTabbedTextExtent (HDC DC, LPSTR Str, int Count, int TabPositions, LPINT TabStopPositions) ;

現在の選択フォントでの *Str* の高さと幅をピクセル単位で計測します。タブの展開を行なう場合は指定します。

引数 *DC*

デバイスコンテキストハンドル。

Str

テキスト行。

Count

Str の文字数。

TabPositions

TabStopPositions のタブストップ数。0 にすればタブは平均 8 文字分展開。

TabStopPositions

昇順のタブストップ位置をピクセル単位で表わす整数型配列。

戻り値 上位ワードで高さ，下位ワードで幅。

GetTempDrive

定義 char GetTempDrive (char DriveLetter) ;

一時ファイル操作を最適アクセス時間で行なうディスクドライブを取得します。

引数 *DriveLetter*

ディスクドライブ文字，ただし現在のドライブを取得する場合は 0。

戻り値 ディスクドライブ文字。

GetTempFileName

定義 int GetTempFileName (char DriveLetter, LPSTR PrefixString,
WORD Unique, LPSTR TempFileName) ;

ルートディレクトリまたは *temp* 環境変数で指定したディレクトリをパス名とする一時ファイルの一意名を取得します。

引数 *DriveLetter*

選択ドライブ，または選択ドライブと ***TF_FORCEDRIVE*** の OR をとった値，デフォルトドライブの場合は 0。

PrefixString

ファイル名の 3 文字プレフィクス（ヌルで終わる文字列)。

Unique

数値のベースファイル名，システム予約値の場合は 0。

TempFileName

144 バイト以上のパス名格納バッファ。

戻り値 数値による一意なファイル名。

GetTextAlign

定義 WORD GetTextAlign (HDC DC) ;

テキスト配置フラグを取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 次のテキスト配置フラグの組み合わせ(第7章の「**TA_テキスト配置オプション**」を参照)。

TA_LEFT, TA_CENTER, TA_RIGHT, TA_BASELINE, TA_BOTTOM, TA_TOP, TA_NOUPDATECP, TA_UPDATECP

関連項目 ***TextOut, ExtTextOut***

GetTextCharacterExtra

定義 int GetTextCharacterExtra (HDC DC) ;

行に書き込まれた文字の間隔を論理単位で取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 現在の文字間隔。

関連項目 ***TextOut, ExtTextOut***

GetTextColor

定義 DWORD GetTextColor (HDC DC) ;

文字の描画に現在使用中のフォアグラウンド色を取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 RGB カラー値。

関連項目 ***TextOut, ExtTextOut***

GetTextExtent

定義 DWORD GetTextExtent (HDC DC, LPSTR Str, int Count) ;

現在の選択フォントでの *Str* の寸法を計算します。

引数 *DC*

デバイスコンテキストハンドル。

Str

テキスト行。

Count

Str の文字数。

戻り値 上位ワードで高さ，下位ワードで幅（それぞれ論理単位)。

GetTextFace

定義 int GetTextFace (HDC DC, int Count, LPSTR FaceName) ;

選択したフォント名を *FaceName* にコピーします。

引数 *DC*

デバイスコンテキストハンドル。

Count

FaceName のサイズ。

FaceName

格納するバッファ。

戻り値 実際にコピーされたバイト数。

GetTextMetrics

定義 BOOL GetTextMetrics (HDC DC, LPTEXTMETRIC Metrics) ;

現在の選択フォントのメトリックスを *Metrics* に取得します。

引数 *DC*

デバイスコンテキストハンドル。

Metrics

格納する ***TEXTMETRIC*** 構造体。

戻り値 成功すれば 0 以外，失敗すれば 0。

GetThresholdEvent

定義 LPINT GetThresholdEvent () ;

最新の限界値イベントを取得します。

戻り値 限界値イベントのポインタ。

GetThresholdStatus

定義 int GetThresholdStatus () ;

限界値イベントステータスを取得します。限界値にまだ達しない音声キューレベルが，各ビットによって表わされます。

戻り値 現在の限界値イベントステータスフラグ。

GetTickCount

定義 DWORD GetTickCount () ;

システム起動後の経過時間を取得します。

戻り値 経過時間（ミリ秒単位)。

GetTopWindow

定義　HWND GetTopWindow (HWND Wnd) ;

最上位の子ウィンドウを取得します。

引数　*Wnd*

親ウィンドウハンドル。

戻り値　子ウィンドウハンドル，ただし該当する子ウィンドウがなければ 0。

GetUpdateRect

定義　BOOL GetUpdateRect (HWND Wnd, LPRECT Rect, BOOL Erase) ;

更新領域を囲む最小矩形を *Rect* に取得します。

引数　*Wnd*

ウィンドウハンドル。

Rect

格納する ***RECT*** 構造体。

Erase

更新領域のバックグラウンドを消去する場合は 0 以外。

戻り値　更新領域が空でない場合は 0 以外，空であれば 0。

関連項目　***WM_ERASEBKGND***

GetUpdateRgn

定義 int GetUpdateRgn (HWND Wnd, HRGN Rgn, BOOL Erase) ;

更新領域を *Rgn* にコピーします。

引数 *Wnd*

ウィンドウハンドル。

Rgn

格納する更新領域。

Erase

バックグラウンドを消去して，子ウィンドウの再描画を行なう場合は 0 以外。

戻り値 ***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION*** のいずれかの領域タイプ（第 7 章の「**Region（領域）フラグ**」を参照)。

GetVersion

定義 WORD GetVersion () ;

Windows の現在のバージョン番号を取得します。

戻り値 上位バイトでバージョン番号の従番号部分，下位バイトで主番号部分。

GetViewportExt

定義 DWORD GetViewportExt (HDC DC) ;

デバイスコンテキストのビューポート範囲を取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 下位ワードでX軸方向範囲，上位ワードでY軸方向範囲（それぞれデバイス単位）。

GetViewportOrg

定義 DWORD GetViewportOrg (HDC DC) ;

デバイスコンテキストのビューポート原点を取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 下位ワードでX座標，上位ワードでY座標（それぞれデバイス単位）。

GetWindow

定義 HWND GetWindow (HWND Wnd, WORD Cmd) ;

Cmd で指定した関係を持つウィンドウを，*Wnd* で指定したウィンドウに取得します。

引数 *Wnd*

オリジナルウィンドウ。

Cmd
次のいずれかの定数（第７章の「**GW_ウィンドウ取得定数**」を参照）。

GW_CHILD, GW_HWNDFIRST, GW_HWNDLAST, GW_HWNDNEXT, GW_HWNDPREV, GW_OWNER

戻り値 ウィンドウハンドル，ただし該当ウィンドウがないか *Cmd* が無効の場合は 0。

GetWindowDC

定義 HDC GetWindowDC (HWND Wnd) ;

ウィンドウのクライアント領域外を主に描画するディスプレイコンテキストを取得します。

引数 *Wnd*
ウィンドウハンドル。

戻り値 ディスプレイコンテキストハンドル，ただしエラーの場合は 0。

関連項目 ***ReleaseDC***

GetWindowExt

定義 DWORD GetWindowExt (HDC DC) ;

ウィンドウの範囲を取得します。

引数 *DC*
デバイスコンテキストハンドル。

戻り値 下位ワードで X 軸方向範囲，上位ワードで Y 軸方向範囲（それぞれ論理単位）。

GetWindowLong

定義 DWORD GetWindowLong (Wnd HWND, int Index) ;

ウィンドウまたはウィンドウクラスの後に割り当てるバイト値に関する情報を取得します。

引数 *Wnd*

ウィンドウハンドル。

Index

バイトオフセット，または ***GWL_EXSTYLE, GWL_STYLE, GWL_WNDPROC*** のいずれかの定数（第 7 章の「**GWL_ウィンドウフィールドオフセット**」を参照）。

戻り値 指定ウィンドウ情報。

GetWindowOrg

定義 DWORD GetWindowOrg (HDC DC) ;

ウィンドウの原点を取得します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 下位ワードで X 論理座標，上位ワードで Y 論理座標。

GetWindowRect

定義 void GetWindowRect (HWND Wnd, LPRECT Rect) ;

境界矩形の寸法を画面座標の ***Rect*** に取得します。

引数 *Wnd*

ウィンドウハンドル。

Rect

格納する ***RECT*** 構造体。

GetWindowsDirectory

定義 WORD GetWindowsDirectory (LPSTR Buffer, int Size) ;

Buffer に Windows ディレクトリのパス名を取得します。

引数 *Buffer*

パス名を格納するバッファ。

Size

最低 144 バイトの格納容量がある *Buffer* サイズ。

戻り値 バッファにコピーされる文字列の長さ。

GetWindowTask

定義 HANDLE GetWindowTask (HWND Wnd) ;

アプリケーションのタスクハンドルを取得します。

引数 *Wnd*

ウィンドウハンドル。

戻り値 タスクハンドル。

GetWindowText

定義 int GetWindowText (HWND Wnd, LPSTR Str, int MaxCount) ;

ウィンドウのタイトル，またはコントロールのテキストを *Str* にコピーします。

引数 *Wnd*

ウィンドウまたはコントロールのハンドル。

Str

文字列を格納するバッファ。

MaxCount

Str のサイズ。

戻り値 実際にコピーされたバイト数，ただしテキストがなければ 0。

GetWindowTextLength

定義 int GetWindowTextLength (HWND Wnd) ;

ウィンドウのタイトル，またはコントロールのテキストの長さを取得します。

引数 *Wnd*

ウィンドウまたはコントロールのハンドル。

戻り値 テキスト長。

GetWindowWord

定義　WORD GetWindowWord (HWND Wnd, int Index) ;

ウィンドウまたはウィンドウクラスの後に割り当てるバイト値に関する情報を取得します。

引数　*Wnd*

ウィンドウハンドル。

Index

正のバイトオフセット，または ***GWW_APPLICATION*** -> ***HINSTANCE, GWW_HWNDPARENT, GWW_ID*** のいずれかの定数（第 7 章の「**GWW_ウィンドウフィールドオフセット**」を参照）。

戻り値　***WORD*** 型値。

GetWinFlags

定義　DWORD GetWinFlags () ;

Windows が動作するメモリコンフィギュレーションフラグを取得します。

戻り値　現在のメモリコンフィギュレーションを指定するフラグマスク。次の定数のいずれか（第 7 章の「**WF_Windows メモリコンフィギュレーションフラグ**」を参照）。

WF_CPU286, WF_CPU386, WF_WIN286, WF_WIN386, WF_LARGEFRAME, WF_PMODE, WF_SMALLFRAME

GlobalAddAtom

定義 ATOM GlobalAddAtom (LPSTR Str) ;

Str をアトムテーブルに加えて，グローバルアトムを新たに作成します。

引数 *Str*
ヌルで終わる文字列。

戻り値 新たに作成されたアトム，ただしエラーの場合は 0。

GlobalAlloc

定義 HANDLE GlobalAlloc (WORD Flags, DWORD Bytes) ;

グローバルヒープから要求メモリ量を最低限割り当てます。

引数 *Flags*
フラグマスク。下記定数のいずれか 1 つ以上（第 7 章の「**GMEM_グローバルメモリフラグ**」を参照)。

GMEM_DDESHARE, GMEM_DISCARDABLE, GMEM_FIXED, GMEM_MOVEABLE, GMEM_NOCOMPACT, GMEM_NODISCARD, GMEM_NOT_BANKED, GMEM_NOTIFY, GMEM_ZEROINIT

Bytes
割り当てたバイト数。

戻り値 割り当てられたグローバルメモリハンドル，ただしエラーの場合は 0。

関連項目 ***GlobalSize***

GlobalCompact

定義 DWORD GlobalCompact (DWORD MinFree) ;

グローバルメモリを圧縮します。場合によっては廃棄可能セグメントを除去して，*MinFree* サイズのブロックを作成します。

引数 *MinFree*
必要なフリーバイト数，ただし廃棄可能セグメントのすべてを廃棄して最大フリーセグメントを取得する場合は 0。

戻り値 フリーブロックの最大サイズ。

GlobalDeleteAtom

定義 ATOM GlobalDeleteAtom (ATOM AnAtom) ;

グローバルアトムの参照カウントを 1 つ減らします。参照カウントが 0 になると，アトムテーブルからアトムと対応文字列が削除されます。

引数 *AnAtom*
ATOM ハンドル。

戻り値 成功すれば 0, 失敗すれば *AnAtom*。

GlobalDiscard

定義 HANDLE GlobalDiscard (HANDLE Mem) ;

指定されたグローバルメモリブロックが廃棄可能でロック解除されていれば，破棄します。

引数 *Mem*

破棄するグローバルメモリブロックのハンドル。

戻り値 成功すればブロックハンドル，失敗すれば 0。

GlobalDosAlloc

定義 DWORD GlobalDosAlloc (DWORD Bytes) ;

リアルモードで動作している MS-DOS からアクセスできるグローバルメモリにメモリブロックを割り当てます。このメモリブロックは，リニアアドレススペースの最初の 1 メガバイト内にグローバルメモリを確保されます。低位メモリは充分な空き領域があるとは限らないので，この関数の使用はできるだけ避けてください。

引数 *Byte*

確保するバイト数。

戻り値 メモリが確保されなければ 0。確保されれば，上位ワードでパラグラフセグメント値，下位ワードでセレクタが返されます。確保したメモリに対するアクセスは，リアルモードではパラグラフセグメント値を，プロテクトモードではセレクタを使って行ないます。

GlobalDosFree

定義 WORD GlobalDosFree (WORD Selector) ;

GlobalDosAlloc によって配置されたメモリブロックを解放します。

引数 *Bytes*

解放されるメモリのセレクタ。

戻り値 成功すれば 0，失敗すれば *Selector*。

GlobalFindAtom

定義　ATOM GlobalFindAtom (LPSTR Str) ;

Str に対応するグローバルアトムを取得します。

引数　*Str*

検索文字列（ヌルで終わる文字列）。

戻り値　グローバルアトム，ただしテーブルに *Str* がなければ 0。

GlobalFix

定義　void GlobalFix (HANDLE Mem) ;

グローバルブロックをメモリに固定するとともに，ロックカウントを 1 つ増やします。

引数　*Mem*

グローバルメモリブロックハンドル。

関連項目　***GlobalUnfix***

GlobalFlags

定義　WORD GlobalFlags (HANDLE Mem) ;

Mem に関する情報を取得します。

引数　*Mem*

グローバルメモリブロックハンドル。

戻り値　上位バイトで ***GMEM_DDESHARE, GMEM_DISCARDABLE, GMEM_DISCARDED, GMEM_NOT_BANKED***，下位バイトでロックカウント（第7章の「**GMEM_グローバルメモリフラグ**」を参照)。

GlobalFree

定義　HANDLE GlobalFree (HANDLE Mem) ;

ロック解除されたグローバルメモリを解放してハンドルを無効にします。

引数　*Mem*

グローバルメモリブロックハンドル。

戻り値　成功すれば 0，失敗すれば *Mem*。

GlobalGetAtomName

定義　WORD GlobalGetAtomName (ATOM AnAtom, LPSTR Buffer, int Size) ;

AnAtom に対応する文字列を *Buffer* にコピーします。

引数　*AnAtom*

アトムハンドル。

Buffer

格納するバッファ。

Size

バッファサイズ。

戻り値　実際にコピーされたバイト数。*AnAtom* が無効であれば 0。

GlobalHandle

定義 DWORD GlobalHandle (WORD Mem) ;

指定セグメントアドレスを持つグローバルメモリクラスのハンドルを取得します。

引数 *Mem*

セグメントアドレス。

戻り値 下位ワードでハンドル，上位ワードでセグメントアドレス。ただし該当ハンドルがなければ 0。

GlobalLock

定義 LPSTR GlobalLock (HANDLE Mem) ;

グローバルメモリブロックの参照カウントをインクリメントしてポインタを返します。

引数 *Mem*

グローバルメモリブロックハンドル。

戻り値 成功すればメモリブロックアドレス，失敗すれば NULL。

GlobalLRUNewest

定義 HANDLE GlobalLRUNewest (HANDLE Mem) ;

廃棄の危険性を最小にするために，グローバルメモリクラスをメモリの LRU（least-recently-used）位置の最後へ移動させます。

引数 *Mem*

グローバルメモリクラスハンドル。

戻り値 無効な *Mem* の場合は 0。

GlobalLRUOldest

定義 HANDLE GlobalLRUOldest (HANDLE Mem) ;

廃棄の可能性を最大にするために，グローバルメモリクラスをメモリの LRU (least-recently-used) 位置の最初へ移動します。

引数 *Mem*

グローバルメモリオブジェクトハンドル。

戻り値 無効な *Mem* の場合は 0。

GlobalNotify

定義 void GlobalNotify (FARPROC NotifyProc) ;

NotifyProc を呼び出して，廃棄するグローバルメモリブロックのハンドルを渡します。*NotifyProc* が 0 以外を返せば，ブロックは廃棄されます。

引数 *NotifyProc*

通知コールバック手続きのインスタンスアドレス。

GlobalPageLock

定義 WORD GlobalPageLock (HANDEL Selector) ;

メモリブロックのページロックカウントをインクリメントします。ロック操作は連続できます。

引数 *Selector*
メモリセレクタ。

戻り値 成功すればインクリメントされたページロックカウント，失敗すれば0。

関連項目 ***GlobalPageUnlock***

GlobalPageUnlock

定義 WORD GlobalPageUnlock (HANDLE Selector) ;

メモリブロックのページロックカウントをデクリメントします。デクリメントされて0になれば，ページの移動またはディスクへのページアウトが可能になります。

引数 *Selector*
メモリセレクタ。

戻り値 成功すればデクリメントされたページロックカウント，失敗すれば0。

関連項目 ***GlobalPageLock***

GlobalReAlloc

定義 HANDLE GlobalReAlloc (HANDLE Mem, DWORD Bytes, WORD Flags) ;

グローバルメモリブロックを ***Bytes*** サイズに再割り当てします。

引数 *Mem*

グローバルメモリブロックハンドル。

Bytes

Mem の新しいサイズ。

Flags

次の定数を1つ以上（第7章の「**GMEM_グローバルメモリフラグ**」を参照)。

GMEM_DISCARDABLE, GMEM_NOTIFY, GMEM_MOVEABLE, GMEM_NOCOMPACT, GMEM_NODISCARD, GMEM_ZEROINIT

戻り値 再割り当てされたグローバルメモリブロックハンドル，ただしエラーの場合は0。

GlobalSize

定義 DWORD GlobalSize (HANDLE Mem) ;

グローバルメモリブロックの現在のサイズを取得します。

引数 *Mem*

グローバルメモリブロックハンドル。

戻り値 実際のサイズ（バイト単位)。*Mem* が無効または廃棄の場合は0。

GlobalUnfix

定義 BOOL GlobalUnfix (HANDLE Mem) ;

GlobalFix がロックしたグローバルメモリブロックのロックを解除します。ブロックのロックカウントが0になれば，ブロックの移動または廃棄が可能です。

引数 *Mem*

グローバルメモリブロックハンドル。

戻り値 ロックカウントが0にデクリメントされれば0, そうでなければ0以外。

GlobalUnlock

定義 BOOL GlobalUnlock (HANDLE Mem) ;

GlobalLock がロックしたグローバルメモリブロックのロックを解除します。ブロックのロックカウントが0になれば，ブロックの移動または廃棄が可能です。

引数 *Mem*

グローバルメモリブロックハンドル。

戻り値 ロックカウントが0にデクリメントされれば0, そうでなければ0以外。

GlobalUnWire

定義 BOOL GlobalUnWire (HANDLE Mem) ;

GlobalWire がロックしたメモリセグメントのロックを解除します。

引数 *Mem*

セグメントハンドル。

戻り値 セグメントのロックが解除されれば 0 以外，そうでなければ 0。

GlobalWire

定義 LPSTR GlobalWire (HANDLE Mem) ;

長期ロックを要するセグメントを低位のメモリに移動してロックします。

引数 *Mem*

セグメントハンドル。

戻り値 成功すれば新しいセグメント位置，失敗すれば NULL。

関連項目 ***GlobalUnWire***

GrayString

定義 BOOL GrayString (HDC DC, HBRUSH Brush, FARPROC OutputFunc,
DWORD Data, int Count, int X, int Y, int Width,
int Height) ;

現在の選択フォントで淡色表示のテキストを描画します。*OutputFunc* を呼び出して，*Data*,*Count*，およびビットマップの高さが *Height* で幅が *Width* の *DC* を渡します。

引数 *DC*

デバイスコンテキストハンドル。

Brush

淡色表示用 ***HBRUSH***。

OutputFunc

描画関数の手続きインスタンスアドレス，ただし ***TextOut*** を使用する場合は NULL。

Data

OutputFunc に渡されるデータ，ただし *OutputFunc* が 0 の場合は文字列。

Count

Data サイズ。0 にすると *Data* は文字列として文字数を計算，-1 にすると *OutputFunc* は 0 を返しイメージは表示されるが判別不能。

X, Y

文字列を囲む矩形の開始位置とする論理位置。

Width

文字列を囲む矩形の幅（論理単位)。0 にすると *Data* を文字列として幅を計算。

Height

文字列を囲む矩形の高さ（論理単位)。0 にすると *Data* を文字列として高さを計算。

戻り値 成功すれば 0 以外，出力関数の返した値が 0 またはビットマップ作成用メモリが不足の場合は 0。

関連項目 ***GetSysColor, SetTextColor, COLOR_GRAYTEXT, MM_TEXT***

HIBYTE

定義 BYTE HIBYTE (WORD A) ;

16 ビット整数値の上位バイトを取り出すマクロです。

引数 *A*

16 ビット整数。

戻り値 上位バイト。

HideCaret

定義 void HideCaret (HWND Wnd) ;

キャレットを表示画面から破壊せずに除去します。

引数 *Wnd*

キャレットを所有するウィンドウ，ただしオーナーウィンドウが現在タスク実行中の場合は 0。

関連項目 ***ShowCaret***

HiliteMenuItem

定義 BOOL HiliteMenuItem (HWND Wnd, HMENU Menu, WORD IDHilite,
WORD Hilite) ;

最上位メニュー項目をハイライト表示，またはハイライトの中止を行ないます。

引数 *Wnd*

ウィンドウハンドル。

Menu

最上位メニューハンドル。

IDHilite

整数型 ID またはメニュー項目の位置。

Hilite

MF_BYCOMMAND か ***MF_BYPOSITION*** と ***MF_HILITE*** か ***MF_UNHILITE*** との組み合わせ（第 7 章の「**MF_メニューフラグ**」を参照）。

戻り値 成功すれば 0 以外，失敗すれば 0。

HIWORD

定義 WORD HIWORD (DWORD AnInteger) ;

32 ビット整数値の上位ワードを抜き出すマクロです。

引数 *AnInteger*

32 ビット整数値。

戻り値 上位ワード。

InflateRect

定義 void InflateRect (LPRECT Rect, int X, int Y) ;

Rect の幅と高さを変更します。矩形の左右端に *X* を加え，上下に *Y* を加えます。

引数 *Rect*

RECT 構造体。

X

矩形の幅を変更する正または負の値。

Y

矩形の高さを変更する正または負の値。

InitAtomTable

定義 BOOL InitAtomTable (int Size) ;

アトムハッシュテーブルを初期化してサイズをセットします。デフォルトのサイズは 37 です。

引数 *Size*

アトムハッシュテーブルのエントリ数（素数）。

戻り値 成功すれば 0 以外，失敗すれば 0。

InSendMessage

定義 BOOL InSendMessage () ;

現在のウィンドウ関数による処理メッセージが，***SendMessage*** 呼び出しによる転送メッセージかどうかを判定します。

戻り値 ***SendMessage*** による転送メッセージの場合は 0 以外，そうでなければ 0。

InsertMenu

定義 BOOL InsertMenu (HMENU Menu, WORD Position, WORD Flags, WORD IDNewItem, LPSTR NewItem) ;

Flags で状態を指定したメニュー項目を新たに挿入します。

引数 *Menu*

メニューハンドル。

Position

挿入される新メニュー項目の前に位置する項目のコマンドIDまたは位置，ただし最後に付加する場合は-1。

Flags

MF_BYCOMMAND または ***MF_BYPOSITION*** と次の定数のいずれかとの組み合わせ（第7章の「**MF_メニューフラグ**」を参照）。

MF_BITMAP, MF_STRING, MF_OWNERDRAW, MF_SEPARATOR, MF_POPUP, MF_MENUBARBREAK, MF_MENUBREAK, MF_CHECKED, MF_UNCHECKED

IDNewItem

新メニュー項目コマンドID，ただしポップアップの場合はメニューハンドル。

NewItem

新メニュー項目の内容。

戻り値 成功すれば0以外，失敗すれば0。

関連項目 ***DrawMenuBar, WM_DRAWITEM, WM_MEASUREITEM***

IntersectClipRect

定義 int IntersectClipRect (HDC DC, int X 1, int Y 1, int X 2, int Y 2) ;

領域と指定矩形の重複部分から新クリッピング領域を作成します。

引数 *DC*

デバイスコンテキストハンドル。

X 1, Y 1

矩形の左上隅。

X 2, Y 2

矩形の右下隅。

戻り値 ***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION*** のいずれか（第7章の「**Region（領域）フラグ**」を参照)。

IntersectRect

定義 int IntersectRect (LPRECT DestRect, LPRECT Src 1 Rect,
LPRECT Src 2 Rect) ;

2つの矩形の重複部分を作成します。

引数 *DestRect*

結果としてできた矩形を表わす ***RECT*** 構造体。

Src 1 Rect

ソース矩形を表わす ***RECT*** 構造体。

Src 2 Rect

ソース矩形を表わす ***RECT*** 構造体。

戻り値 重複部分があれば0以外，なければ0。

InvalidateRect

定義 void InvalidateRect (HWND Wnd, LPRECT Rect, BOOL Erase) ;

ウィンドウの更新領域に *Rect* を加えて，クライアント領域を無効にします。

引数 *Wnd*

ウィンドウハンドル。

Rect

更新領域に加える ***RECT*** 構造体，ただしクライアント領域全体の場合は NULL。

Erase

BeginPaint によるバックグラウンド消去の場合は 0 以外。

関連項目 ***ValidateRect, ValidateRgn, WM_PAINT***

InvalidateRgn

定義 void InvalidateRgn (HWND Wnd, HRGN Rgn, BOOL Erase) ;

ウィンドウの更新領域に *Rgn* を加えて，クライアント領域を無効にします。

引数 *Wnd*

ウィンドウハンドル。

Rgn

領域ハンドル（クライアント座標系)。

Erase

BeginPaint によるバックグラウンド消去の場合は 0 以外。

関連項目 ***ValidateRect, ValidateRgn, WM_PAINT***

InvertRect

定義　void InvertRect (HDC DC, LPRECT Rect) ;

Rect で指定した矩形の中のすべての色を反転します。

引数　*DC*

デバイスコンテキストハンドル。

Rect

RECT 構造体（論理座標系）。

InvertRgn

定義　BOOL InvertRgn (HDC DC, HRGN Rgn) ;

Rgn で指定した領域の中のすべての色を反転します。

引数　*DC*

デバイスコンテキストハンドル。

Rgn

領域ハンドル（デバイス単位)。

戻り値　成功すれば 0 以外，失敗すれば 0。

IsCharAlpha

定義　BOOL IsCharAlpha (char AChar) ;

言語ドライバと現在の言語によって *AChar* が英字であるかどうかを判定します。

引数　*AChar*

判定する文字。

戻り値 英字であれば 0 以外，英字でなければ 0。

IsCharAlphaNumeric

定義 BOOL IsCharAlphaNumeric (char AChar) ;

言語ドライバと現在の言語によって *AChar* が英数字であるかどうかを判定します。

引数 *AChar*
判定する文字。

戻り値 英数字であれば 0 以外，英数字でなければ 0。

IsCharLower

定義 BOOL IsCharLower (char AChar) ;

言語ドライバと現在の言語によって *AChar* が小文字であるかどうかを判定します。

引数 *AChar*
判定する文字。

戻り値 小文字であれば 0 以外，小文字でなければ 0。

IsCharUpper

定義 BOOL IsCharUpper (char AChar) ;

言語ドライバと現在の言語によって *AChar* が大文字であるかどうかを判定します。

引数 *AChar*

判定する文字。

戻り値 大文字であれば 0 以外，大文字でなければ 0。

IsChild

定義 BOOL IsChild (HWND Parent, HWND Wnd) ;

Wnd が *Parent* の子ウィンドウであるかどうかを判定します。

引数 *Parent*

ウィンドウハンドル。

Wnd

判定するウィンドウ。

戻り値 子ウィンドウであれば 0 以外，子ウィンドウでなければ 0。

IsClipboardFormatAvailable

定義 BOOL IsClipboardFormatAvailable (WORD Format) ;

指定書式のデータがクリップボードにあるかどうかを判定します。

引数 *Format*

登録済みのクリップボード書式で，***CF*_**定数のいずれか（第 7 章の「**CF_クリ**

ップボード書式」を参照)。

戻り値 *Format* にデータがあれば0以外，なければ0。

関連項目 ***SetClipboardData***

IsDialogMessage

定義 BOOL IsDialogMessage (HWND Dlg, LPMSG Msg) ;

モードなしダイアログボックスのメッセージを判定して処理します。その際に，キーボードメッセージはコマンドメッセージに変換されます。

引数 *Dlg*
ダイアログボックスハンドル。

Msg
MSG 構造体。

戻り値 ***TranslateMessage*** と ***DispatchMessage*** を呼び出さずに *Msg* が処理されれば0以外，処理されなければ0。

IsDlgButtonChecked

定義 WORD IsDlgButtonChecked (HWND Dlg, int IDButton) ;

ボタンコントロールのチェックの有無を判定します。

引数 *Dlg*
ダイアログボックスハンドル。

IDButton
ボタンコントロールID。

戻り値 チェックがあれば0以外，なければ0。3状態ボタンの場合は淡色表示が2，チェックあれば1，チェックがなければ0。

IsIconic

定義 BOOL IsIconic (HWND Wnd) ;

ウィンドウがアイコン形状（最小化）かどうかを判定します。

引数 ***Wnd***

ウィンドウハンドル。

戻り値 最小化されていれば0以外，それ以外は0。

IsRectEmpty

定義 BOOL IsRectEmpty (LPRECT Rect) ;

矩形の幅，高さが0かどうかを判定します。

引数 ***Rect***

RECT 構造体。

戻り値 空であれば0以外，空でなければ0。

IsWindow

定義 BOOL IsWindow (HWND Wnd) ;

Wnd が既存の有効ウィンドウかどうかを判定します。

引数 ***Wnd***

ウィンドウハンドル。

戻り値 有効であれば0以外，無効であれば0。

IsWindowEnabled

定義 BOOL IsWindowEnabled (HWND Wnd) ;

ウィンドウがマウスとキーボードからの入力を受けつけられるかどうかを判定します。

引数 *Wnd*

ウィンドウハンドル。

戻り値 有効であれば 0 以外，無効であれば 0。

IsWindowVisible

定義 BOOL IsWindowVisible(HWND Wnd) ;

ShowWindow によるウィンドウ表示が行なわれているどうかを判定します。

引数 *Wnd*

ウィンドウハンドル。

戻り値 画面上にあれば（オーバーラップされて完全に見えない場合でも）0 以外，なければ 0。

IsZoomed

定義 BOOL IsZoomed (HWND Wnd) ;

ウィンドウが最大化されているかどうかを判定します。

引数 *Wnd*

ウィンドウハンドル。

戻り値 最大化されていれば 0 以外，そうでなければ 0。

KillTimer

定義 BOOL KillTimer (HWND Wnd, int IDEvent) ;

タイマーイベントを止めて，メッセージキューから対応する ***WM_TIMER*** メッセージを除去します。

引数 *Wnd*

ウィンドウハンドル。

IDEvent

タイマイベント ID。

戻り値 成功すれば 0 以外，無効な *IDEvent* の場合は 0。

関連項目 ***SetTimer***

_lclose

定義 int _lclose (int FileHandle) ;

指定ファイルをクローズします。

引数 *FileHandle*

DOS ファイルハンドル。

戻り値 成功すれば 0, 失敗すれば - 1。

_lcreat

定義 int _lcreat (LPSTR PathName, int Attribute) ;

指定ファイルをオープンします。

引数 *PathName*

オープンするファイルの完全な DOS パス名。

Attribute

読み書き両用は 0, 読み込み専用は 1, 非表示は 2, システムファイルは 3。

戻り値 成功すれば DOS ファイルハンドル，失敗すれば -1。

LimitEmsPages

定義 void LimitEmsPages (DWORD Kbytes) ;

拡張メモリを使って動作するアプリケーションに割り当てる拡張メモリのキロバイト数を制限します。

引数 *Kbytes*

キロバイト数。

LineDDA

定義 void LineDDA (int X 1, int Y 1, int X 2, int Y 2, FARPROC LineFunc,
LPSTR Data) ;

ライン上の連続座標位置をすべて算出し，***LineFunc*** を呼び出して座標位置の ***X*** 座標と ***Y*** 座標，***Data*** を渡します。

引数 *X 1, Y 1*

ライン上の最初の座標位置。

X 2, Y 2

ライン上の最後の座標位置。

LineFunc

コールバック関数の手続きインスタンスアドレス。

Data

LineFunc に渡されるデータ。

LineTo

定義 BOOL LineTo (HDC DC, int X, int Y) ;

現在の位置から指定の終点までの線を選択されたペンで描きます。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

線の終点。

戻り値 描ければ 0 以外，描けなければ 0。

_llseek

定義 DWORD _llseek (int FileHandle, DWORD Offset, int Origin) ;

オープンファイルにポインタを位置づけます。

引数 *FileHandle*

DOS ファイルハンドル。

Offset

ポインタを移動するバイト数。

Origin

原点と移動方向の指定。ファイルの先頭から前方への移動は 0, 現在の位置からの移動は 1, ファイルの末尾から逆戻りする移動は 2。

戻り値 ポインタの新しいオフセット，ただし失敗すれば - 1。

LoadAccelerators

定義 HANDLE LoadAccelerators (HANDLE Instance, LPSTR TableName) ;

実行形式ファイルから指定された名前のアクセラレータテーブルをロードします。

引数 *Instance*

実行形式ファイルにアクセラレータテーブルを含むモジュールインスタンス。

TableName

アクセラレータテーブル名（ヌルで終わる文字列），または整数型 ID。

戻り値 成功すればアクセラレータテーブルハンドル，失敗すれば 0。

LoadBitmap

定義 HBITMAP LoadBitmap (HANDLE Instance, LPSTR BitmapName) ;

指定された名前のビットマップリソースをロードします。

引数 *Instance*

実行形式ファイルにビットマップを含むモジュールインスタンス，ただし定義済みのビットマップの場合は 0。

BitmapName

ヌルで終わる文字列，ビットマップを指定する整数型 ID，***OBM***_定数が指定する定義済みのビットマップ（第 7 章の「**OBM_定義済みビットマップ**」を参照）。

戻り値 成功すればビットマップハンドル，失敗すれば 0。

LoadCursor

定義　HCURSOR LoadCursor (HANDLE Instance, LPSTR CursorName) ;

指定された名前のカーソルリソースをロードします。

引数　*Instance*

実行形式ファイルにカーソルを含むモジュールインスタンス，ただし定義済みのカーソルの場合は 0。

CursorName

ヌルで終わる文字列，整数型 ID，***IDC***_定数が指定する定義済みカーソルのいずれか（第 7 章の「**IDC_標準カーソル ID**」を参照)。

戻り値　成功すればカーソルハンドル，該当カーソルがなければ 0, カーソルリソースがなければ未定義。

LoadIcon

定義　HICON LoadIcon (HANDLE Instance, LPSTR IconName) ;

指定された名前のアイコンリソースをロードします。

引数　*Instance*

実行形式ファイルにアイコンを含むモジュールインスタンス，ただし定義済みのアイコンの場合には 0。

IconName

文字列，整数型 ID，***IDI***_定数が指定する定義済みのアイコンのいずれか（第 7 章の「**IDI_標準アイコン ID**」を参照)。

戻り値　成功すればアイコンハンドル，失敗すれば 0。

LoadLibrary

定義 HANDLE LoadLibrary (LPSTR LibFileName) ;

指定された名前のライブラリモジュールをロードします。

引数 *LibFileName*

ライブラリファイル名（ヌルで終わる文字列)。

戻り値 成功すれば値が 32 を超えるライブラリモジュールインスタンスハンドル，失敗すれば値は 32 以下となり，次のいずれかの値。

メモリなしの場合は 0, タスクに対してリンクしようとしたときは 5, マルチプルデータセグメントの場合は 6, 無効な Windows バージョンの場合は 10, 無効な EXE ファイルの場合は 11, OS/2 アプリケーションの場合は 12, DOS 4.0 アプリケーションの場合は 13, 無効な EXE タイプの場合は 14, プロテクトモードで，Windows の以前のバージョンに対して作成された.EXE をロードしようとした場合は 15, MULTIPLE 属性（各インスタンスに 1 つのセグメント）と READWRITE 属性（書き込み可能）を持つ.EXE の 2 番目のインスタンスをロードしようとした場合は 16, ラージフレーム EMS モードで，すでに使っている共有不可能な DLL にリンクするアプリケーションの 2 番目のインスタンスをロードしようとした場合は 17, リアルモードで，プロテクトモード専用としてマークされたアプリケーションをロードしようとした場合は 18。

LoadMenu

定義 HMENU LoadMenu (HANDLE Instance, LPSTR MenuName) ;

指定した名前のメニューリソースをロードします。

引数 *Instance*

実行形式ファイルにメニューを含むモジュールインスタンス。

MenuName

ヌルで終わる文字列，またはメニューの整数型 ID 名。

戻り値　成功すればメニューハンドル，失敗すれば0。

LoadMenuIndirect

定義　HMENU LoadMenuIndirect (LPSTR MenuTemplate) ;

MenuTemplate に基づきメニューをロードします。

引数　*MenuTemplate*

MENUITEMTEMPLATE 構造体の配列。

戻り値　成功すればメニューハンドル，失敗すれば0。

LoadModule

定義　HANDLE LoadModule (LPSTR ModuleName, LPSTR ParameterBlock) ;

Windows アプリケーションをロードして実行します。

引数　*ModuleName*

アプリケーションファイル名（ヌルで終わる文字列）。

ParameterBlock

4フィールド構造体。

WORD　*EnvSeg* は環境セグメントアドレス，ただし Windows 環境の場合は0。
DWORD *CmdLine* はコマンド行。
DWORD　*CmdShow* は2ワード構造体で1番目のワードが2に設定，2番目のワードが ***ShowWindow*** の *CmdShow* に設定。
DWORD *Reserved* は常に0。

戻り値　***LoadLibrary*** に同じ。

関連項目　***WinExec***

LoadResource

定義 HANDLE LoadResource (HANDLE Instance, HANDLE ResInfo) ;

メモリを割り当ててリソースをロードします。

引数 *Instance*

実行形式ファイルにリソースを含むモジュールインスタンス。

ResInfo

FindResource によって返されるリソースハンドル。

戻り値 成功すればリソースハンドル，失敗すれば0。

関連項目 ***LockResource***

L
Fun

LoadString

定義 int LoadString (HANDLE Instance, WORD ID, LPSTR Buffer,
int BufferMax) ;

指定された名前の文字列リソースをロードし，ヌル文字を末尾につけて ***Buffer*** にコピーします。

引数 *Instance*

実行形式ファイルに文字列を含むモジュールインスタンス。

ID

文字列の整数型 ID。

Buffer

格納するバッファ。

BufferMax

Buffer サイズ。

戻り値 実際にコピーされるバイト数，リソースがなければ0。

LOBYTE

定義 BYTE LOBYTE (WORD A) ;

16 ビット整数値の下位バイトを取り出すマクロです。

引数 *A*
16 ビット整数。

戻り値 下位バイト。

LocalAlloc

定義 HANDLE LocalAlloc (WORD Flags, WORD Bytes) ;

ローカルヒープからローカルメモリブロックを割り当てます。実際のサイズは指定サイズを上回ることがあります。

引数 *Flags*
1 つ以上の次の定数（第 7 章の「**LMEM_ローカルメモリフラグ**」を参照)。
LMEM_DISCARDABLE, LMEM_FIXED, LMEM_MODIFY, LMEM_MOVEABLE, LMEM_NOCOMPACT, LMEM_NODISCARD, LMEM_ZEROINIT

Bytes
割り当てられたブロックのサイズ（バイト単位)。

戻り値 成功すればローカルメモリブロックハンドル，失敗すれば 0。

関連項目 ***LockData***

LocalCompact

定義 LocalCompact (WORD MinFree) : WORD ;

Minfree を最低サイズとする空きメモリブロックを生成します。必要であれば，ロック解除状態のブロックを関数によって移動または廃棄します。

引数 *MinFree*

必要な空きメモリのバイト数，ただし手近の空きメモリブロックから最大のものを取得するだけの場合は 0。

戻り値 最大ブロックのサイズ（バイト数）。

LocalDiscard

定義 HANDLE LocalDiscard (HANDLE Mem) ;

指定されたローカルメモリブロックを破棄します。ハンドルは有効なまま ***LocalReAlloc*** を呼び出せば再使用できます。

引数 *Mem*

ローカルメモリブロックへのハンドル。

戻り値 成功すれば 0，失敗すれば *Mem*。

LocalFlags

定義 WORD LocalFlags (HANDLE Mem) ;

メモリブロックに関する情報を取得します。

引数 *Mem*

ローカルメモリブロックハンドル。

戻り値 上位バイトで ***LMEM_DISCARDABLE*** または ***LMEM_DISCARDED***, 下位バイトでロックカウント (第7章の「**LMEM_ローカルメモリフラグ**」を参照)。

LocalFree

定義 HANDLE LocalFree (HANDLE Mem) ;

ローカルメモリブロックを解放してハンドルを無効にします。

引数 *Mem*

ローカルメモリブロックハンドル。

戻り値 成功すれば 0, 失敗すれば *Mem*。

LocalHandle

定義 HANDLE LocalHandle (WORD Mem) ;

指定アドレスにあるローカルメモリクラスのハンドルを取得します。

引数 *Mem*

ローカルメモリクラスアドレス。

戻り値 ローカルメモリクラスハンドル。

LocalInit

定義 BOOL LocalInit (WORD Segment, WORD Start, WORD End) ;

ローカルヒープを初期化して ***GlobalLock*** を呼び出し，セグメントをロックします。

引数 *Segment*

ローカルヒープのセグメントアドレス。

Start

ローカルヒープの先頭のオフセットアドレス。

End

ローカルヒープの末尾のオフセットアドレス。

戻り値 初期化されれば 0 以外，失敗すれば 0。

L
Fun

LocalLock

定義 LPSTR LocalLock(HANDLE Mem) ;

Mem をロックしてロックカウントをインクリメントします。ブロックの移動または廃棄はできません。

引数 *Mem*

ローカルメモリブロックハンドル。

戻り値 成功すればブロックへのポインタ，失敗すれば NULL。

LocalReAlloc

定義 HANDLE LocalReAlloc (HANDLE Mem, WORD Bytes, WORD Flags) ;

ローカルメモリブロックのサイズまたは属性を *Flags* の指定どおりに変更します。

引数 *Mem*

ローカルメモリブロックハンドル。

Bytes

ブロックの新しいサイズ（バイト数)。

Flags

1つ以上の次の定数（第7章の「**LMEM_ローカルメモリフラグ**」を参照)。

LMEM_DISCARDABLE, LMEM_MODIFY, LMEM_MOVEABLE, LMEM_NOCOMPACT, LMEM_NODISCARD, LMEM_ZEROINIT

戻り値 成功すればローカルメモリブロックハンドル，失敗すれば0。

関連項目 ***LockData***

LocalShrink

定義 WORD LocalShrink (HANDLE Seg, WORD Size) ;

ローカルヒープを指定サイズに縮小します。ただし，アプリケーションのモジュール定義ファイルに指定される最小サイズを下回らないようにします。

引数 *Seg*

ローカルヒープを含むセグメント，ただし現データセグメントの場合は0。

Size

指定サイズ（バイト数)。

戻り値 縮小後のサイズ。

関連項目 ***GlobalSize***

LocalSize

定義 WORD LocalSize (HANDLE Mem) ;

ローカルメモリブロックの現在のサイズを取得します。

引数 *Mem*

ローカルメモリブロックハンドル。

戻り値 バイト単位のブロックサイズ，無効な *Mem* の場合は 0。

L
Fu

LocalUnlock

定義 BOOL LocalUnlock (HANDLE Mem) ;

ロックカウントをデクリメントしローカルメモリブロックのロックを解除します。

引数 *Mem*

ローカルメモリブロックハンドル。

戻り値 ロックカウントがデクリメントされて 0 になりロックが移動または廃棄の対象になれば 0, そうでなければ 0 以外。

LockData

定義 HANDLE LockData (int Dummy) ;

移動可能データセグメントをメモリ内にロックします。

引数 *Dummy*

使いません（0を設定）。

戻り値 成功すればロックされたデータセグメントのハンドル，失敗すれば0。

関連項目 ***UnLockData***

LockResource

定義 LPSTR LockResource (HANDLE ResData) ;

ロードされているリソースのアドレスを取得してロックカウントをインクリメントします。リソースは移動または廃棄の対象とはなりません。

引数 *ResData*

LoadResource が返すリソースハンドル。

戻り値 ロードされているリソースへのポインタ，リソースがなければNULL。

関連項目 ***UnlockResource***

LockSegment

定義 HANDLE LockSegment (WORD Segment) ;

セグメントをロックしてロックカウントをインクリメントします。ただし，プロテクトモードの廃棄不能セグメントは除きます。

引数 *Segment*

セグメントアドレス，ただし現在のセグメントの場合は - 1。

戻り値 セグメントへのポインタ，エラーまたは廃棄の場合は NULL。

関連項目 ***UnlockSegment***

_lopen

定義 int _lopen (LPSTR PathName, int ReadWrite) ;

指定ファイルをオープンします。

引数 *PathName*

パス名とファイル名を指定する文字列。

ReadWrite

OF_READ, OF_READWRITE, OF_WRITE などの ***OF***_定数を使用して読み書きのアクセス方法を指定します（第 7 章の「**OF_ファイルオープン定数**」を参照)。

戻り値 成功すれば DOS ファイルハンドル，失敗すれば - 1。

LOWORD

定義 WORD LOWORD (DWORD AnInteger) ;

32 ビット整数値の下位ワードを抽出します。

引数 *AnInteger*

32 ビット整数。

戻り値 下位ワード。

LPtoDP

定義 BOOL LPtoDP (HDC DC, LPPOINT Points, int Count) ;

現在のマッピングモードの ***Points*** の論理座標系をデバイス座標系に変換します。

引数 *DC*

デバイスコンテキストハンドル。

Points

POINT 構造体の配列。

Count

Points のサイズ。

戻り値 座標位置すべて変換されれば 0 以外，そうでなければ 0。

_lread

定義 WORD _lread (int FileHandle, LPSTR Buffer, int Bytes) ;

オープンファイルからバイト数を読み込みます。

引数 *FileHandle*

DOS ファイルハンドル。

Buffer

格納するバッファ。

Bytes

読み込むバイト数。

戻り値 成功すれば読み込みバイト数，失敗すれば - 1, ファイルの終わりの場合は 0。

lstrcat

定義　LPSTR lstrcat (LPSTR Str 1, LPSTR Str 2) ;

Str1 に *Str2* を連結します。

引数　*Str1*

最初の文字列（ヌルで終わる文字列)。

Str2

2 番目の文字列（ヌルで終わる文字列)。

戻り値　成功すれば *Str1*, 失敗すれば 0。

lstrcmp

定義　int lstrcmp (LPSTR Str 1, LPSTR Str 2) ;

現在選択しているフォントに基づいて，2 つの文字列の辞書的な文字配列の比較を大小文字を区別して行ないます。小文字の方が大文字より大きい値とされます。

引数　*Str1*

第 1 の文字列（ヌルで終わる文字列)。

Str2

第 2 の文字列（ヌルで終わる文字列)。

戻り値　*Str1* < *Str2* であれば 0 未満，*Str1* = *Str2* であれば 0, *Str1* > *Str2* であれば 0 を超える値。

lstrcmpi

定義　int lstrcmpi (LPSTR Str 1, LPSTR Str 2) ;

現在選択されているフォントに基づいて，2つの文字列の辞書的な文字配列の比較を大小文字を区別せずに行ないます。

引数　*Str1*

第1の文字列（ヌルで終わる文字列）。

Str2

第2の文字列（ヌルで終わる文字列）。

戻り値　*Str1* < *Str2* であれば0未満，*Str1* = *Str2* であれば0, *Str1* > *Str2* であれば0を超える値。

lstrcpy

定義　LPSTR lstrcpy (LPSTR Str 1, LPSTR Str 2) ;

ヌル文字も含めて *Str2* を *Str1* にコピーします。

引数　*Str1*

第1の文字列（ヌルで終わる文字列）。

Str2

第2の文字列（ヌルで終わる文字列）。

戻り値　成功すれば *Str1* へのポインタ，失敗すれば0。

lstrlen

定義 int lstrlen (LPSTR Str) ;

Str のヌル文字以外の長さを計算します。

引数 *Str*

ヌルで終わる文字列。

戻り値 *Str* の長さ（バイト単位)。

_lwrite

定義 WORD _lwrite (int FileHandle, LPSTR Buffer, int Bytes) ;

バッファから指定ファイルへデータを書き込みます。

引数 *FileHandle*

DOS ファイルハンドル。

Buffer

書き込みデータを含むバッファ。

Bytes

書き込むバイト数。

戻り値 成功すればファイルに書き込まれたバイト数，失敗すれば - 1。

MAKEINTATOM

定義 LPSTR MAKEINTATOM (WORD Integer) ;

10 進数の文字列を表わす整数アトムを作成するマクロです。作成されたアトムは，***AddAtom*** でアトムテーブルに追加することができます。

引数 *Interger*

アトムの文字列とする数値。

戻り値 作成されたアトムへのポインタ。

MAKEINTRESOURCE

定義 LPSTR MAKEINTRESOURCE (int Integer) ;

整数値を文字列への long 型ポインタに整数値を変換するマクロです。long 型ポインタの上位ワードは 0 に設定されます。

引数 *Interger*

変換対象の整数値。

戻り値 文字列へのポインタ。

MAKELONG

定義 DWORD MAKELONG (WORD Low, WORD High) ;

2 つの ***WORD*** 値を連結して 1 つの unsigned long 値を作成するマクロです。

引数 *Low*

新しい unsigned long 値の下位ワード。

High
新しい unsigned long 値の上位ワード。

戻り値 作成された符号なしの ***DWORD***。

MAKEPOINT

定義 POINT MAKEPOINT (DWORD Integer) ;

座標を示す long 値を ***POINT*** 構造体へ変換するマクロです。

引数 *AnInteger*
点の x 座標と y 座標を指定します。

戻り値 ***POINT*** 構造体。

M
Fu

MakeProcInstance

定義 FARPROC MakeProcInstance (FARPROC Proc, HANDLE Instance) ;

指定エクスポート関数の手続きインスタンスアドレスを作成します。

引数 *Proc*
エクスポート関数 ***FARPROC*** のアドレス。

Instance
モジュールインスタンスハンドル。

戻り値 成功すれば関数の手続きインスタンスアドレス，失敗すれば 0。

MapDialogRect

定義 void MapDialogRect (HWND Dlg, LPRECT Rect) ;

Rect のダイアログボックスの単位を画面単位に変換します。

引数 *Dlg*

ダイアログボックスハンドル。

Rect

RECT 構造体。

関連項目 ***GetDialogBaseUnits***

MapVirtualKey

定義 WORD MapVirtualKey (WORD Code, WORD MapType) ;

仮想キーまたはキー走査コードを，*MapType* で指定した対応する走査コード，仮想キーコード，ASCII 値に割り当てます。

引数 *Code*

MapType 値によって決定される仮想キーまたはキー走査コード。

MapType

0 は仮想キーコード，1 は走査コード，2 は仮想キーコード。

戻り値 *MapType* が 0 であれば走査コード，1 であれば仮想キーコード，2 であればシフトなしの ASCII 値。

MessageBeep

定義 void MessageBeep (WORD BeepType) ;

システムのスピーカーからビープ音を発生させます。

引数 *BeepType*

0 に設定。

MessageBox

定義 int MessageBox (HWND Parent, LPSTR Txt, LPSTR Caption,
WORD TextType) ;

指定メッセージとタイトル，さらに *TextType* で指定した定義済みのアイコンとプッシュボタンを含むダイアログボックスを作成して表示します。

引数 *Parent*

メッセージボックスを所有するウィンドウ。

Txt

表示メッセージ（ヌルで終わる文字列）。

Caption

ダイアログボックスタイトル（ヌルで終わる文字列），ただし“エラー”の場合はNULL。

TextType

*MB*_定数，または，その組み合わせ（第7章の「**MB_メッセージボックスフラグ**」を参照）。

戻り値 メモリ不足の場合は0，成功すれば次のいずれか（第7章の「**ID_ダイアログボックスコマンドID**」を参照）。

ID_ABORT, ID_CANCEL, ID_IGNORE, ID_NO, ID_OK, ID_RETRY, ID_YES

ModifyMenu

定義 BOOL ModifyMenu (HMENU Menu, WORD Position, WORD Flags,
WORD IDNewItem, LPSTR NewItem) ;

既存のメニュー項目を変更して Flags で指定した新しい状態にします。

引数 *Menu*

メニューハンドル。

Position

コマンド ID またはメニュー項目位置。

Flags

MF_BYCOMMAND か ***MF_BYPOSITION*** と下記定数の組み合わせ（第7章の「**MF_メニューフラグ**」を参照)。

MF_DISABLED, MF_ENABLED, MF_GRAYED, MF_BITMAP, MF_STRING, MF_OWNERDRAW, MF_SEPARATOR, MF_POPUP, MF_MENUBARBREAK, MF_MENUBREAK, MF_CHECKED, MF_UNCHECKED

IDNewItem

コマンド ID，ただし *Flags* の設定が ***MF_POPUP*** の場合は変更メニュー項目のメニューハンドル。

NewItem

文字列(***MF_STRING***), ***HBITMAP*** (***MF_BITMAP***), またはアプリケーション提供のデータ（***MF_OWNERDRAW***)。

戻り値 成功すれば 0 以外，失敗すれば 0。

MoveTo

定義 DWORD MoveTo (HDC DC, int X, int Y) ;

現在の位置を指定座標位置に移動します。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

新しい位置。

戻り値 下位ワードで旧位置の X 座標，上位ワードで旧位置の Y 座標。

MoveWindow

定義 void MoveWindow (HWND Wnd, int X, int Y, int Width, int Height, BOOL Repaint) ;

WM_SIZE メッセージをウィンドウに送ります。***WM_SIZE*** で渡される幅と高さの値はクライアント領域のものです。

引数 *Wnd*

ポップアップウィンドウまたは子ウィンドウのハンドル。

X, Y

ウィンドウの新しい左上隅。

Width

新しいウィンドウの幅。

Height

新しいウィンドウの高さ。

Repaint

移動後にウィンドウを再描画する場合は 0 以外。

MulDiv

定義 int MulDiv (int Number, int Numerator, int Denominator) ;

Numerator に *Number* を掛けて，その値を *Denominator* で割ります。結果の値は四捨五入されます。

引数 *Number*
値。

Numerator
値。

Denominator
値。

戻り値 結果値。桁あふれや *Denominator* が 0 の場合は 32767 または－32767。

NetBIOSCall

定義 NetBIOSCall ;

NETBIOS 割り込み INT 5 Ch を呼び出します。これからのバージョンとの互換性を保つには，INT 5 Ch ソフトウェア割り込みへの直接呼び出さず，この関数を使ってください。

割り込み呼び出しには CPU レジスタを設定しなければならないので，この関数を呼び出すのはアセンブリ言語ルーチンからに限ってください。

OemKeyScan

定義 DWORD OemKeyScan (WORD OemChar) ;

OemChar を OEM 走査コードに割り当てます。

引数 *OemChar*

OEM ASCII 文字コード（0 - 0xFF)。

戻り値 成功すれば下位ワードで走査コード，上位ワードでシフト状態(ビット 2 は ***CTRL***, ビット 1 は ***SHIFT*** キーが押されている)，失敗すれば下位ワードと上位ワードで - 1。

OemToAnsi

定義 BOOL OemToAnsi (LPSTR OemStr, LPSTR AnsiStr) ;

OemStr を ANSI 文字セットに変換します。

引数 *OemStr*

OEM 文字列（ヌルで終わる文字列)。

AnsiStr

格納するバッファ，または *OemStr* 変換位置。

戻り値 常に偽（- 1)。

OemToAnsiBuff

定義 void OemToAnsiBuff (LPSTR OemStr, LPSTR AnsiStr, int Length) ;

OemStr を ANSI 文字セットに変換します。

引数 *OemStr*

OEM 文字のバッファ。

AnsiStr

格納するバッファ，または *OemStr* 変換位置。

Length

OemStr の長さ。

OffsetClipRgn

定義 int OffsetClipRgn (HDC DC, int X, int Y) ;

X 軸と Y 軸の指定オフセットにしたがってデバイスのクリッピング領域を移動します。

引数 *DC*

デバイスコンテキストハンドル。

X

左右方向の移動論理単位。

Y

上下方向の移動論理単位。

戻り値 ***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION*** のいずれか（第 7 章の「**Region（領域）フラグ**」を参照)。

OffsetRect

定義 void OffsetRect (LPRECT Rect, int X, int Y) ;

X 軸方向と Y 軸方向の指定オフセットにしたがって矩形の座標を調節します。

引数 *Rect*

RECT 構造体。

X

左右方向の移動単位。

Y

上下方向の移動単位。

OffsetRgn

定義 int OffsetRgn (HRGN Rgn, int X, int Y) ;

X 軸方向と Y 軸方向の指定オフセットにしたがって領域を移動します。

引数 *Rgn*

領域ハンドル。

X

左右方向の移動単位。

Y

上下方向の移動単位。

戻り値 ***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION*** のいずれか（第 7 章の「**Region（領域）フラグ**」を参照)。

OffsetViewportOrg

定義 DWORD OffsetViewportOrg (HDC DC, int X, int Y) ;

現在の原点に X 軸方向と Y 軸方向の指定値を加算してビューポート原点を変更します。

引数 *DC*

デバイスコンテキストハンドル。

X

X 座標の原点オフセット。

Y

Y 座標の原点オフセット。

戻り値 下位ワードで直前の原点の X 座標，上位ワードで直前の原点の Y 座標。

OffsetWindowOrg

定義 DWORD OffsetWindowOrg (HDC DC, int X, int Y) ;

現在の原点に X 軸方向と Y 軸方向の指定値を加算してウィンドウの原点を変更します。

引数 *DC*

デバイスコンテキストハンドル。

X

X 座標の原点オフセット（論理単位）。

Y

Y 座標の原点オフセット（論理単位）。

戻り値 上位ワードで直前の原点の X 座標，下位ワードで直前の原点の Y 座標。

OpenClipboard

定義 BOOL OpenClipboard (HWND Wnd) ;

クリップボードをオープンして，他のアプリケーションが内容を変更するのを防ぎます。

引数 *Wnd*

ウィンドウハンドル。

戻り値 成功すれば 0 以外，他アプリケーションによるオープン済みの場合は 0。

関連項目 ***CloseClipboard***

OpenComm

定義 OpenComm (LPSTR ComName, WORD InQueue, WORD OutQueue) ;

通信デバイスをオープンします。デバイスはデフォルトの構成に初期化され，送受信キューが割り当てられます。

引数 *ComName*

n を整数値とする"COM*n*"または"LPT*n*"を含む文字列。

InQueue

受信キューサイズ，ただし LPT ポートの場合には無視。

OutQueue

送信キューサイズ，ただし LPT ポートの場合には無視。

戻り値 成功すれば *Cid* ハンドル，失敗すれば次のいずれかの負のエラー値（第 7 章の「**IE_通信オープンエラーフラグ**」を参照)。

IE_BADID, IE_BAUDRATE, IE_BYTESIZE, IE_DEFAULT, IE_HARDWARE, IE_MEMORY, IE_NOPEN, IE_OPEN

関連項目 ***SetCommState***

OpenFile

定義 int OpenFile (LPSTR FileName, LPOFSTRUCT ReOpenBuff, WORD Style) ;

ファイルの作成，オープン，再オープン，削除を行ないます。

引数 *FileName*

指定ファイル名。

ReOpenBuff

オープン時にファイルの情報を格納する ***OFSTRUCT*** 構造体。

Style

動作を指定する ***OF***_定数のいずれか（第 7 章の「**OF_ファイルオープン定数**」を参照)。

戻り値 成功すれば DOS ファイルハンドル，失敗すれば - 1。

OpenIcon

定義　BOOL OpenIcon (HWND Wnd) ;

最小化されたウィンドウを元のサイズと位置に復元します。

引数　*Wnd*
ウィンドウハンドル。

戻り値　成功すれば 0 以外，失敗すれば 0。

OpenSound

定義　int OpenSound () ;

演奏デバイスをオープンして，他のアプリケーションにオープンされないようにします。

戻り値　使用可能な音声数，使用中の場合は ***S_SERDVNA***，メモリ不足の場合は ***S_SEROFM***（第 7 章の「**S_サウンド定数**」を参照）。

OutputDebugString

定義　void OutputDebugString (LPSTR OutputString) ;

デバッガがあればデバッガへ，なければ Windows のデバッグバージョンの AUX ポートに *OutputString* を送ります。

引数　*OutputString*
ヌルで終わる文字列。

PaintRgn

定義 BOOL PaintRgn (HDC DC, HRGN Rgn) ;

選択されたブラシで領域を塗りつぶします。

引数 *DC*

デバイスコンテキストハンドル。

Rgn

塗りつぶす領域。

戻り値 成功すれば 0 以外，失敗すれば 0。

PALETTEINDEX

定義 COLORREF PALETTEINDEX (int PaletteIndex) ;

パレットエントリのインデックスを受け取り，戻り値として上位バイトが 1 で，下位バイトがパレットエントリインデックスの値を返すマクロです。明示的な RGB 値のかわりに使います。

引数 *PaletteIndex*

色を格納したパレットエントリのインデックス。

戻り値 論理パレットインデックス ID。

PALETTERGB

定義 DWORD PALETTERGB (BYTE Red, BYTE Green, BYTE Blue) ;

輝度が 0 から 255 までの 3 原色値から，パレット相対 RGB 値を生成するマクロです。戻り値は上位バイトが 2 になります。

P Func

引数 *Red*

赤の輝度値。

Green

緑の輝度値。

Blue

青の輝度値。

戻り値 生成された RGB 値。

PatBlt

定義 BOOL PatBlt (HDC DC, int X, int Y, int Width, int Height, DWORD Rop) ;

Rop を実行して，選択したブラシとデバイスの既存パターンでビットパターンを作成します。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

矩形の左上隅。

Width

矩形の幅（論理単位)。

Height

矩形の高さ（論理単位)。

Rop

次のいずれかのラスタ操作コード（第 7 章の「**Ternary Raster Operation(3 次元ラスタ操作)**」を参照)。

PATCOPY, PATINVERT, DSTINVERT, BLACKNESS, WHITENESS

戻り値 ビットパターンが描ければ 0 以外，描けなければ 0。

PeekMessage

定義 BOOL PeekMessage (LPMSG Msg, HWND Wnd, WORD MsgFilterMin, WORD MsgFilterMax, WORD RemoveMsg) ;

アプリケーションのキューを調べてメッセージがあれば ***Msg*** にコピーします。メッセージがなければただちに復帰して制御権を Windows に譲ります。

引数 *Msg*

格納する ***MSG*** 構造体。

Wnd

メッセージ送り先ウィンドウ，アプリケーションに所属するウィンドウに送る場合は 0, ***PostAppMessage*** 関数によって送られるメッセージの場合は - 1。

MsgFilterMin

調査対象の最下位メッセージ ID，調査対象を限定しない場合は 0。

MsgFilterMax

調査対象の再上位メッセージ ID，調査対象を限定しない場合は 0。

RemoveMsg

PM_NOREMOVE, PM_NOYIELD, PM_REMOVE のいずれか，または組み合わせ（第 7 章の「**PM_ピークメッセージオプション**」を参照)。

戻り値 メッセージがあれば 0 以外，なければ 0。

関連項目 ***GetMessage, WaitMessage***

Pie

定義 BOOL Pie (HDC DC, int X 1, int Y 1, int X 2, int Y 2, int X 3, int Y 3, int X 4, int Y 4) ;

矩形の中央に選択したペンによってパイ形の円弧を描き，その内部を選択したブラシで塗りつぶします。

引数 *DC*

デバイスコンテキストハンドル。

X 1, Y 1

矩形の左上隅。

X 2, Y 2

矩形の左下隅。

X 3, Y 3

円弧の始点。

X 4, Y 4

円弧の終点。

戻り値 描ければ 0 以外，描けなければ 0。

PlayMetaFile

定義 BOOL PlayMetaFile (HDC DC, HANDLE MF) ;

指定デバイス上でメタファイルの内容を描画します。

引数 *DC*

デバイスコンテキストハンドル。

MF

メタファイルハンドル。

戻り値 成功すれば 0 以外，失敗すれば 0。

PlayMetaFileRecord

定義 void PlayMetaFileRecord (HDC DC, LPHANDLETABLE HandleTable,
LPMETAHEADER MetaRecord, WORD Handles) ;

メタファイルレコードに含まれる GDI 関数呼び出しを実行します。

引数 *DC*

デバイスコンテキストハンドル。

HandleTable

メタファイル再描画用の ***HANDLETABLE***。

MetaRecord

実行するメタファイルの ***METARECORD***。

Handles

HandleTable のサイズ。

関連項目 ***EnumMetaFile***

Polygon

定義 BOOL Polygon (HDC DC, LPPOINT Points, int Count) ;

現在の多角形塗りつぶしモードで，*Points* で指定した各頂点を結ぶ多角形を描きます。場合によっては閉鎖多角形にします。

引数 *DC*

デバイスコンテキストハンドル。

Points

POINT 構造体の配列。

Count

Points の頂点数。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***SetPolyFillMode***

Polyline

定義 BOOL Polyline (HDC DC, LPPOINT Points, int Count) ;

選択されたペンによって一連の線分を描きます。線分の各接点は *Points* によって指定されます。

引数 *DC*

デバイスコンテキストハンドル。

Points

POINT 構造体の配列。

Count

配列のポイント数。

戻り値 成功すれば 0 以外，失敗すれば 0。

PolyPolygon

定義 BOOL PolyPolygon (HDC DC, LPPOINT Points, LPINT PolyCounts,
int Count) ;

現在の多角形塗りつぶしモードで，*Points* で指定した頂点を結ぶ多角形を複数描きます。その際，多角形がたがいにオーバーラップする場合もあります。多角形を自動的にクローズすることはできません。

引数 *DC*

デバイスコンテキストハンドル。

Points

POINT 構造体の配列。

PolyCounts

Points で指定した各多角形の頂点の数を指定する整数型配列。

Count

多角形の数。

戻り値 描ければ0以外，描けなければ0。

PostAppMessage

定義 BOOL PostAppMessage (HANDLE Task, WORD Msg, WORD wParam, DWORD lParam) ;

アプリケーションにメッセージを送ります。メッセージの ***Wnd*** の値は0にセットされます。

引数 *Task*

メッセージの送り先アプリケーション。

Msg

メッセージ種別。

wParam

付加メッセージ情報。

lParam

付加メッセージ情報。

戻り値 成功すれば0以外，失敗すれば0。

関連項目 ***GetCurrentTask, GetMessage, PeekMessage***

PostMessage

定義 BOOL PostMessage (HWND Wnd, WORD Msg, WORD wParam,
DWORD lParam) ;

アプリケーションのウィンドウにメッセージを送ります。

引数 *Wnd*

メッセージの送り先ウィンドウ，ただしオーバーラップウィンドウとポップアップウィンドウすべてに送る場合は 0xFFFF。

Msg

メッセージ種別。

wParam

付加メッセージ情報。

lParam

付加メッセージ情報。

戻り値 成功すれば 0 以外，失敗すれば 0。

PostQuitMessage

定義 void PostQuitMessage (int ExitCode) ;

主に ***WM_DESTROY*** メッセージへの応答として ***WM_QUIT*** メッセージを送ります。

引数 *ExitCode*

アプリケーション終了コードで， ***WM_QUIT*** メッセージの *wParam*。

PtInRect

定義　BOOL PtInRect (LPRECT Rect, POINT Point) ;

座標位置が矩形の内部，左辺，上辺のいずれかにあるかどうかを判定します。

引数　*Rect*

RECT 構造体。

Point

POINT 構造体。

戻り値　*Point* 位置が *Rect* 内部の場合は 0 以外，そうでなければ 0。

PtInRegion

定義　BOOL PtInRegion (HRGN Rgn, int X, int Y) ;

座標位置が指定領域内にあるかどうかを判定します。

引数　*Rgn*

領域ハンドル。

X, Y

座標位置。

戻り値　*Point* 位置が *Rgn* 内部の場合は 0 以外，そうでなければ 0。

PtVisible

定義 BOOL PtVisible (HDC DC, int X, int Y) ;

座標位置が指定デバイスのクリッピング領域内にあるかどうかを判定します。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

座標位置。

戻り値 *DC* のクリッピング領域内に *Point* がある場合は 0 以外，そうでなければ 0。

ReadComm

定義 int ReadComm (int Cid,LPSTR Buf, int Size) ;

Cid を読み込みます。*Size* で指定した文字数が *Buf* にコピーされます。

引数 *Cid*

通信デバイス。

Buf

格納するバッファ。

Size

Buf サイズ。

戻り値 実際に読み込まれた文字数，受信キューに文字がなければ 0, エラーの場合は負の値。

関連項目 ***GetCommError***, ***OpenComm***

RealizePalette

定義 WORD RealizePalette (HDC DC) ;

指定した論理パレットをシステムパレットのエントリに割り当てます。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 最後に実現したパレットにおいて，システムパレットエントリに新たに割り当てられた論理パレットのエントリ数。

Rectangle

定義 BOOL Rectangle (HDC DC, int X 1, int Y 1, int X 2, int Y 2) ;

現在の選択ペンによって矩形を描き，内部を現在の選択ブラシで塗りつぶします。

引数 *DC*

デバイスコンテキストハンドル。

X 1, Y 1

矩形の左上隅。

X 2, Y 2

矩形の右下隅。

戻り値 矩形が描ければ 0 以外，失敗すれば 0。

RectInRegion

定義 BOOL RectInRegion (HRGN Region, LPRECT Rect) ;

指定領域内に *Rect* が収まっているかどうかを判定します。

引数 *Region*

HRGN 領域ハンドル。

Rect

RECT 構造体。

戻り値 領域内にあれば0以外，領域外であれば0。

RectVisible

定義 BOOL RectVisible (HDC DC, LPRECT Rect) ;

デバイスコンテキストの指定クリッピング領域内に，*Rect* があるかどうかを判定します。

引数 *DC*

デバイスコンテキストハンドル。

Rect

RECT 構造体。

戻り値 クリッピング領域内にあれば0以外，外であれば0。

RegisterClass

定義 BOOL RegisterClass (LPWNDCLASS WndClass) ;

WndClass で指定した属性を持つウィンドウクラスを登録します。ウィンドウク

ラスの登録は 1 回限りです。

引数 *WndClass*

WNDCLASS 構造体。

戻り値 クラスが登録されれば 0 以外，登録されなければ 0。

RegisterClipboardFormat

定義 WORD RegisterClipboardFormat (LPSTR FormatName) ;

クリップボード書式を登録します。書式を登録すると参照カウントがインクリメントされます。

引数 *FormatName*

書式名（ヌルで終わる文字列）。

戻り値 成功すれば 0xC000 から 0xFFFF までの登録書式ハンドル，失敗すれば 0。

RegisterWindowMessage

定義 WORD RegisterWindowMessage (LPSTR Str) ;

システム全体のウィンドウメッセージを一意に定義します。

引数 *Str*

登録する文字列（ヌルで終わる文字列）。

戻り値 成功すれば 0xC000 から 0xFFFF までのメッセージハンドル，失敗すれば 0。

ReleaseCapture

定義　void ReleaseCapture () ;

マウスを解放して通常の入力処理に戻ります。

関連項目　***SetCapture***

ReleaseDC

定義　int ReleaseDC (HWND Wnd, HDC DC) ;

通常デバイスコンテキスト，またはウィンドウデバイスコンテキストを解放して，他アプリケーションによる使用を可能にします。ウィンドウクラスやプライベートウィンドウに変化はありません。

引数　*Wnd*

ウィンドウハンドル。

DC

デバイスコンテキストハンドル。

戻り値　デバイスコンテキストが解放されれば 1, 解放されなければ 0。

関連項目　***GetDC, GetWindowDC***

RemoveFontResource

定義　BOOL RemoveFontResource (LPSTR FileName) ;

Windows フォントテーブルからフォントを除去します。除去前に，該当フォントに対する未処理の参照をすべて削除します。

引数 *FileName*

フォントリソースファイル名(ヌルで終る文字列),またはモジュールインスタンスハンドル。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***DeleteObject, WM_FONTCHANGE***

RemoveMenu

定義 BOOL RemoveMenu (HMENU Menu, WORD Position, WORD Flags) ;

指定メニューから項目と対応ポップアップを破壊せずに除去します。除去後のポップアップは再使用できます。

引数 *Menu*

メニューハンドル。

Position

コマンド ID，またはメニュー項目位置。

Flags

Position 引数の特性を指定する ***MF_BYCOMMAND*** または ***MF_BYPOSITION***（第 7 章の「**MF_メニューフラグ**」を参照)。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***DrawMenuBar, GetSubMenu***

RemoveProp

定義 HANDLE RemoveProp (HWND Wnd, LPSTR Str) ;

ウィンドウのプロパティリストから，*Str* で指定したエントリを除去します。返されたデータハンドルはアプリケーション側で解放してください。

引数 *Wnd*

ウィンドウハンドル。

Str

ヌルで終わる文字列，またはアトム。

戻り値 文字列へのデータハンドル，ただし該当文字列がなければ 0。

関連項目 ***AddAtom***

ReplyMessage

定義 void ReplyMessage (DWORD Reply) ;

SendMessage 呼び出しによる送信メッセージに応答します。引き続き ***SendMessage*** の呼び出しと ***ReplyMessage*** の応答を実行することができます。

引数 *Reply*

メッセージ依存の応答。

ResizePalette

定義 BOOL ResizePalette (HPALETTE Palette, WORD NumEntries) ;

論理パレットサイズを *NumEntries* に変更します。パレットサイズが拡大される場合の追加エントリは，黒にセットされます。

引数 *Palette*

論理パレットハンドル。

NumEntries

新しいパレットサイズ。

戻り値 成功すれば 0 以外，失敗すれば 0。

RestoreDC

定義 BOOL RestoreDC (HDC DC, int SaveDC) ;

デバイスコンテキストを *SaveDC* で指定した元の状態にコンテキストスタックからコピーして復元します。*SaveDC* がスタック最上部にないと状態情報は削除されてしまいます。

引数 *DC*

デバイスコンテキストハンドル。

SaveDC

前回の *SaveDC* 呼び出しの戻り値。最終保管デバイスコンテキストの場合は - 1。

戻り値 復元されれば 0 以外，復元されなければ 0。

RGB

定義 DWORD RGB (BYTE Red, BYTE Green, BYTE Blue) ;

輝度範囲が 0 から 255 の 3 原色値から RGB カラー値を生成をするマクロです。

引数 *Red*

赤の輝度値。

Green

緑の輝度値。

Blue

青の輝度値。

戻り値 生成された RGB 値。

RoundRect

定義　BOOL RoundRect (HDC DC, int X 1, int Y 1, int X 2, int Y 2, int X 3, int Y 3) ;

選択したペンによって4隅の丸い矩形を描き，選択したブラシで内部を塗りつぶします。

引数　*DC*

デバイスコンテキストハンドル。

X 1, Y 1

矩形の左上隅。

X 2, Y 2

矩形の右下隅。

X 3

丸い角を描くための楕円の幅。

Y 3

丸い角を描くための楕円の高さ。

戻り値　描ければ0以外，描けなければ0。

SaveDC

定義　int SaveDC (HDC DC) ;

DC の現在の状態をコンテキストスタックに保存します。

引数　*DC*

デバイスコンテキストハンドル。

戻り値　成功すれば保存されたデバイスコンテキスト，失敗すれば0。

関連項目　***RestoreDC***

ScaleViewportExt

定義 DWORD ScaleViewportExt (HDC DC, int Xnum, int Xdenom, int Ynum,
int Ydenom) ;

現在のビューポート範囲を変更します。

引数 *DC*

デバイスコンテキストハンドル。

Xnum

現X軸方向範囲を乗ずる値。

Xdenom

現X軸方向範囲を除する値。

Ynum

現Y軸方向範囲を乗ずる値。

Ydenom

現Y軸方向範囲を除する値。

戻り値 上位ワードで変更前のY軸方向範囲，下位ワードで変更前のX軸方向範囲。

ScaleWindowExt

定義 DWORD ScaleWindowExt (HDC DC, int Xnum, int Xdenom,
int Ynum, int Ydenom) ;

現在のウィンドウ範囲を変更します。

引数 *Xnum*

現X軸方向範囲を乗ずる値。

Xdenom

現X軸方向範囲を除する値。

Ynum

現 Y 軸方向範囲を乗ずる値。

Ydenom

現 Y 軸方向範囲を除する値。

戻り値 上位ワードで変更前の Y 軸方向範囲，下位ワードで変更前の X 軸方向範囲。

ScreenToClient

定義 void ScreenToClient (HWND Wnd, LPPOINT Point) ;

Point の画面座標値を変換して，指定ウィンドウのクライアント座標に置き換えます。

引数 *Wnd*

ウィンドウハンドル。

Point

置き換えられる ***POINT*** 構造体。

ScrollDC

定義 BOOL ScrollDC (HDC DC, int dx, int dy, LPRECT Scroll, LPRECT Clip, HRGN UpdateRgn, LPRECT UpdateRect) ;

dx と *dy* 単位分だけ，ビットの矩形をスクロールします。

引数 *DC*

デバイスコンテキストハンドル。

dx

水平スクロール単位。

dy
垂直スクロール単位。

Scroll
スクロール矩形を含む ***RECT*** 構造体。

Clip
クリッピング矩形を含む ***RECT*** 構造体。

UpdateRgn
スクロール後の矩形に含まれない ***ScrollDC*** 領域。NULL の場合は更新領域の計算なし。

UpdateRect
スクロール更新領域の矩形を含む対象 ***RECT*** 構造体。NULL の場合は更新矩形の計算なし。

戻り値 成功すれば 0 以外，失敗すれば 0。

ScrollWindow

定義 void ScrollWindow (HWND Wnd, int XAmount, int YAmount, LPRECT Rect, LPRECT ClipRect) ;

XAmount と *YAmount* の分だけ，クライアント領域をスクロールします。スクロール後の矩形に含まれない領域はウィンドウの更新領域に結合されます。

引数 *Wnd*
ウィンドウハンドル。

XAmount
X 軸方向のスクロール単位（デバイス単位)。

YAmount
Y 軸方向のスクロール単位（デバイス単位)。

Rect
スクロールの対象となるクライアント領域 ***RECT*** 構造体。クライアント領域全体をスクロールする場合は NULL。

ClipRect
スクロールの対象となるクリッピング ***RECT*** 構造体。ウィンドウ全体をスクロールする場合は NULL。

関連項目 ***UpdateWindow, WM_PAINT***

SelectClipRgn

定義 int SelectClipRgn (HDC DC, HRGN Rgn) ;

Rgn のコピーをデバイスコンテキストの現クリッピング領域にします。

引数 *DC*
デバイスコンテキストハンドル。

Rgn
選択対象の領域。

戻り値 ***COMPLEXREGION, ERROR, NULLREGION, SIMPLEREGION*** の領域種別定数のいずれか（第 7 章の「**Region（領域）フラグ**」を参照)。

SelectObject

定義 HANDLE SelectObject (HDC DC, HANDLE Object) ;

DC の論理オブジェクトを選択します。選択するオブジェクトは一度につき 1 つに限ります。オブジェクトは不要になったら削除してください。

引数 *DC*
デバイスコンテキストハンドル。

Object
ビットマップ，ブラシ，フォント，ペン，領域のいずれか。

戻り値 置き換え対象のオブジェクト，メタファイル *DC* の場合は 0 以外，エラーの場合は 0。

関連項目 ***DeleteObject, SelectClipRgn, SelectPalette***

SelectPalette

定義 HPALETTE SelectPalette (HDC DC, HPALETTE Palette,
BOOL ForceBackground) ;

デバイスコンテキストの指定パレットオブジェクトとして *Palette* を選択して，GDI による表示カラーの制御に使用します。

引数 *DC*
デバイスコンテキストハンドル。

Palette
選択対象の論理パレット。

ForceBackground
0 以外であればバックグラウンドパレット。ウィンドウに入力フォーカスがあるときに 0 であれば，フォアグラウンドパレット。

戻り値 成功すれば置き換えられた論理パレット，失敗すれば 0。

関連項目 ***CreatePalette***

SendDlgItemMessage

定義 DWORD SendDlgItemMessage (HWND Dlg, int IDDlgItem, WORD Msg,
WORD wParam, DWORD lParam) ;

IDDlgItem で指定したダイアログボックスコントロールにメッセージを送ります。メッセージ処理後に関数は値を返します。

引数 *Dlg*

ダイアログボックスハンドル。

IDDlgItem

送り先ダイアログ項目の整数型 ID。

Msg

メッセージ種別。

wParam

付加メッセージ情報。

lParam

付加メッセージ情報。

戻り値 コントロールのウィンドウ関数が返す値，ただし無効 *IDDlgItem* の場合は 0。

SendMessage

定義 DWORD SendMessage (HWND Wnd, WORD Msg, WORD wParam,
DWORD lParam) ;

指定ウィンドウのウィンドウ関数にメッセージを送ります。メッセージ処理後に関数は値を返します。

引数 *Wnd*

メッセージの送り先ウィンドウ，ただしシステムのポップアップウィンドウすべてに送る場合は 0xFFFF。

Msg

メッセージ種別。

wParam

追加メッセージ情報。

lParam

追加メッセージ情報。

戻り値 メッセージを受け取ったウィンドウ関数が返す値。

SetActiveWindow

定義 HWND SetActiveWindow (HWND Wnd) ;

最上位ウィンドウをアクティブにします。

引数 *Wnd*

ウィンドウハンドル。

戻り値 直前にアクティブだったウィンドウハンドル。

SetBitmapBits

定義 DWORD SetBitmapBits (HBITMAP Bitmap, DWORD Count, LPSTR Bits) ;

Bitmap のビットを *Bits* のビット値にセットします。

引数 *Bitmap*

セットされる ***HBITMAP***。

Count

バイト数による *Bits* サイズ。

Bits
ビットマップビットの ***BYTE*** 配列。

戻り値 ビットマップビットのセットに要するバイト数，エラーの場合は 0。

SetBitmapDimension

定義 SetBitmapDimension (HBITMAP ABitmap, int X, int Y) ;

ビットマップの高さと幅を 0.1 ミリ単位でセットします。

引数 *Bitmap*
ビットマップハンドル。

X
0.1 ミリ単位のビットマップの幅。

Y
0.1 ミリ単位のビットマップの高さ。

戻り値 上位ワードで直前の高さ，下位ワードで直前の幅。

関連項目 ***GetBitmapDimension***

SetBkColor

定義 DWORD SetBkColor (HDC DC, COLORREF Color) ;

現在のバックグラウンドカラーを *Color* に設定します。またはデバイスがサポートする最も近い物理カラーにします。

引数 *DC*
デバイスコンテキストハンドル。

Color

新バックグラウンド ***COLORREF***。

戻り値 成功すれば以前の RGB バックグラウンド色，失敗すれば 0x80000000。

関連項目 ***BitBlt, SetBkMode, StretchBlt***

SetBkMode

定義 int SetBkMode (HDC DC, int BkMode) ;

バックグラウンドモードを設定します。すなわちテキスト描画にハッチブラシあるいは実線以外のペンスタイルを使用するとき，既存バックグラウンド色を除去するかどうかをあらかじめ指定します。

引数 *DC*

デバイスコンテキストハンドル。

BkMode

OPAQUE または ***TRANSPARENT*** のいずれかのモード（第7章の「**Background Mode（バックグラウンドモード）**」を参照）。

戻り値 成功すれば以前のモード，失敗すれば 0。

SetBrushOrg

定義 DWORD SetBrushOrg (HDC DC, int X, int Y) ;

選択されているブラシの原点を設定します。ストックオブジェクトのブラシは設定できません。

引数 *DC*

デバイスコンテキストハンドル。

X, Y
0から7の範囲内の新しい原点（デバイス単位)。

戻り値 以前の原点（下位ワードでX座標，上位ワードでY座標)。

SetCapture

定義 HWND SetCapture (HWND Wnd) ;

マウスカーソルの位置にかかわらず，マウスからの入力すべてを ***Wnd*** に送ります。

引数 *Wnd*
ウィンドウハンドル。

戻り値 直前の全マウス入力送り先のウィンドウ，該当ウィンドウがなければ0。

関連項目 ***ReleaseCapture***

SetCaretBlinkTime

定義 void SetCaretBlinkTime (WORD MSeconds) ;

キャレット点滅の間隔を設定します。

引数 *MSeconds* ミリ秒単位のブリンク時間。

SetCaretPos

定義 void SetCaretPos (int X, int Y) ;

キャレットを指定位置に移動します。

引数 *X, Y*

新しい位置（論理座標）。

SetClassLong

定義 DWORD SetClassLong (HWND Wnd, int Index, DWORD Newlong) ;

WNDCLASS 構造体の ***Index*** で指定した long 値を置き換えます。

引数 *Wnd*

ウィンドウハンドル。

Index

GCL_MENUNAME, ***GCL_WNDPROC***，または後に割り当てる 4 バイト値をセットする正のバイトオフセット（第 7 章の「**GCL_クラスフィールドオフセット**」を参照)。

NewLong

置き換え値。

戻り値 直前の値。

SetClassWord

定義 WORD SetClassWord (HWND Wnd, int Index, WORD NewWord) ;

WNDCLASS 構造体の *Index* で指定したワード値を置き換えます。

引数 *Wnd*

ウィンドウハンドル。

Index

GCW_CBCLSEXTRA, GCW_CBWNDEXTRA, GCW_HBRBACKGROUND, GCW_HCURSOR, GCW_HICON, GCW_HMODULE, GCW_STYLE，または後に割り当てる2バイト値をセットする正のバイトオフセット（第7章の「**GCW_クラスフィールドオフセット**」を参照)。

NewWord

置き換え値。

戻り値 直前の値。

SetClipboardData

定義 HANDLE SetClipboardData (WORD Format, HANDLE Mem) ;

Format のクリップボードへのデータハンドルを設定します。関数が値を返す前にデータハンドルが解放される場合がほとんどです。

引数 *Format*

CF_クリップボード書式定数のいずれか（第7章の「**CF_クリップボード書式**」を参照)。

Mem

Format のデータを含むグローバルメモリブロックへのハンドル。***WM_RENDERFORMAT*** メッセージの場合は 0。

戻り値 クリップボードが割り当てたデータハンドル。

SetClipboardViewer

定義 HWND SetClipboardViewer (HWND Wnd) ;

ウィンドウチェインにウィンドウを追加して，クリップボードに変更があるたびに ***WM_DRAWCLIPBOARD*** メッセージから通知を受けられるようにします。

引数 *Wnd*

ウィンドウハンドル。

戻り値 クリップボードビューアチェイン内の次のウィンドウハンドル。

関連項目 ***ChangeClipboardChain, WM_CHANGECBCHAIN, WM_DRAWCLIPBOARD, WM_DESTROY***

SetCommBreak

定義 int SetCommBreak (int Cid) ;

文字転送を保留にして，デバイスの転送ラインをブレーク状態にします。

引数 *Cid*

通信デバイス。

戻り値 成功すれば 0, ただし無効な *Cid* の場合は負の値。

関連項目 ***ClearCommBreak, OpenComm***

SetCommEventMask

定義 PWORD SetCommEventMask (int Cid, WORD EvtMask) ;

デバイスのイベントマスクの現在の状態を取得して有効にします。

引数 *Cid*

通信デバイス。

EvtMask

次の定数の組み合わせ（第 7 章の「**EV_通信イベント定数**」を参照)。

EV_BREAK, EV_CTS, EV_DSR, EV_ERR, EV_PERR, EV_RING, EV_RLSD, EV_RXCHAR, EV_RXFLAG, EV_TXEMPTY

戻り値 イベントマスクへのポインタを返します。各ビットのセットでイベントの発生を示します。

関連項目 ***OpenComm***

SetCommState

定義 int SetCommState (DCB DCB) ;

DCB の *Id* フィールドで指定した通信デバイスを再初期化して，*DCB* の指定状態にします。送受信キューに変更はありません。

引数 *DCB*

DCB 構造体。

戻り値 成功すれば 0, 失敗すれば負の値。

SetCursor

定義 HCURSOR SetCursor (HCURSOR Cursor) ;

指定カーソルリソースにカーソル形状を設定します。

引数 *Cursor*

LoadCursor から返されたカーソルリソースハンドル。

戻り値 カーソルの直前の形状，直前のカーソルがなければ 0。

SetCursorPos

定義 void SetCursorPos (int X, int Y) ;

指定画面座標にカーソルを移動します。***ClipCursor*** 矩形外への移動は修正されます。

引数 *X, Y*

カーソルの新しい位置（画面座標）。

SetDIBits

定義 int SetDIBits (HDC DC, HANDLE Bitmap, WORD StartScan, WORD NumScans, LPSTR Bits, LPBITMAPINFO BitsInfo, WORD Usage) ;

デバイス独立のビットマップ（DIB）の形式に合わせて，ビットマップのビットをセットします。

引数 *DC*

デバイスコンテキストハンドル。

Bitmap

ビットマップハンドル。

StartScan

Bits の最初の走査ラインに対応する走査ライン番号。

NumScans

Bits の走査ライン数。

Bits

BitsInfo の ***biBitCount*** フィールドで書式指定した DIB ビットのバイト配列。

BitsInfo

DIB 情報を含む ***BITMAPINFO*** 構造体。

Usage

BitsInfo の ***bmiColors*** フィールドの内容を記述する ***DIB_RGB_COLORS*** または ***DIB_PAL_COLORS***（第 7 章の「**DIB_カラーテーブル識別子**」を参照)。

戻り値 成功すれば実際にコピーされた走査ライン数，失敗すれば 0。

SetDIBitsToDevice

定義 int SetDIBitsToDevice (HDC DC, WORD DestX, WORD DestY, WORD Width,
WORD Height, WORD SrcX, WORD SrcY,
WORD StartScan, WORD NumScans, LPSTR Bits,
LPBITMAPINFO BitsInfo, WORD Usage) ;

デバイス独立のビットマップ（DIB）から直接デバイス表面にビットを設定します。

引数 *DC*

デバイスコンテキストハンドル。

DestX, DestY

デバイス対象矩形原点。

Width

DIB 矩形の X 軸方向範囲。

Height
DIB 矩形の Y 軸方向範囲。

SrcX, SrcY
DIB ソース位置。

StartScan
Bits の最初の走査ラインに対応する DIB 走査ライン数。

NumScans
Bits の DIB 走査ライン数。

Bits
BitsInfo の ***biBitCount*** フィールドで書式指定された DIB ビットのバイト配列。

BitsInfo
DIB 情報を含む ***BITMAPINFO*** 構造体。

Usage
BitsInfo の ***bmiColors*** フィールドの内容を記述した ***DIB_RGB_COLORS*** または ***DIB_PAL_COLORS***（第 7 章の「**DIB_カラーテーブル識別子**」を参照）。

戻り値 設定された走査ライン数。

SetDlgItemInt

定義 void SetDlgItemInt (HWND Dlg, int IDDlgItem, WORD Value,
BOOL Signed) ;

ダイアログボックスコントロールのテキストを，*Value* で指定した変換文字列値にセットします。

引数 *Dlg*
ダイアログボックスハンドル。

IDDlgItem
コントロールの整数型 ID。

Value
セット値。

Signed
Value が符号つきであれば 0 以外。

関連項目 ***WM_SETTEXT***

SetDlgItemText

定義 void SetDlgItemText (HWND Dlg, int IDDlgItem, LPSTR Str) ;

ダイアログボックスのタイトルまたはコントロールのテキストを，*Str* で指定した値に設定します。

引数 *Dlg*
ダイアログボックスハンドル。

IDDlgItem
コントロールの整数型 ID。

Str
ヌルで終わる文字列。

関連項目 ***WM_SETTEXT***

SetDoubleClickTime

定義 void SetDoubleClickTime (WORD Count) ;

マウスによるダブルクリックの最初のクリックと次のクリックの最長間隔を設定します。

引数 *Count*
ダブルクリック間隔を示すミリ秒数，ただしデフォルト値の 500 を使う場合は 0。

SetEnvironment

定義 int SetEnvironment (LPSTR PortName, LPSTR Environ, WORD Count) ;

デバイス環境の作成または置き換えを行ないます。

引数 *PortName*

システムポート名（ヌルで終る文字列)。

Environ

新しい環境を含むバッファ。

Count

コピー対象の *Environ* のバイト数，ただし現在の環境を削除する場合は 0。

戻り値 実際にコピーされたバイト数，エラーの場合は 0, 環境を削除した場合は - 1。

SetErrorMode

定義 BOOL SetErrorMode (WORD Mode) ;

DOS ファンクション 24 H エラー時にエラーメッセージボックスを表示するかどうかを指定します。指定がない場合は，Windows による元の INT 21 H 呼び出しが無効になり，アプリケーションによってエラーが処理されます。

引数 *Mode*

0 の場合はエラーメッセージボックス表示，1 の場合は Windows からアプリケーションに制御権移行。

戻り値 設定されれば 0 以外，セットできなければ 0。

SetFocus

定義 HWND SetFocus (HWND Wnd) ;

ウィンドウに入力フォーカスを割り当て，キーボードからの入力すべてを送ります。

引数 *Wnd*
ウィンドウハンドル，ただしキーストロークを無視する場合は 0。

戻り値 直前に入力フォーカスがあったウィンドウ，ただし該当ウィンドウがなければ 0。

SetHandleCount

定義 WORD SetHandleCount (WORD Number) ;

タスク用のファイルハンドル数を *Number* の指定値に変更します。

引数 *Number*
必要ファイルハンドル数（最大 255）。

戻り値 実際使用可能なファイルハンドル数(*Number* 以下)。

SetKeyboardState

定義 void SetKeyboardState (BYTE KeyState) ;

Windows のキーボード状態テーブルに，*KeyState* をコピーします。

引数 *KeyState*
キー状態を含む 255 バイトの配列。

関連項目 ***GetKeyboardState***

SetMapMode

定義 int SetMapMode (HDC DC, int MapMode) ;

デバイスコンテキストのマッピングモードをセットします。マッピングモードは，GDI の論理単位からデバイス単位への変換と，X 軸と Y 軸の方向を定義します。

引数 *DC*

デバイスコンテキストハンドル。

MapMode

下記のマッピングモード定数のいずれか（第 7 章の「**MM_マッピングモード**」を参照）。

MM_ANISOTROPIC, MM_HIENGLISH, MM_HIMETRIC, MM_ISOTROPIC, MM_LOENGLISH, MM_LOMETRIC, MM_TEXT, MM_TWIPS

戻り値 直前のマッピングモード。

SetMapperFlags

定義 DWORD SetMapperFlags (HDC DC, DWORD Flag) ;

Flag で指定したフォントマッピングアルゴリズムを変更して，論理フォントから物理フォントにマップを行ないます。

引数 *DC*

デバイスコンテキストハンドル。

Flag

最初のビットを 1 にセットすると，デバイスの X と Y のアスペクト比と等しいフォントのみが選択されます。

戻り値 以前のフォントマッピングフラグ。

SetMenu

定義 BOOL SetMenu (HWND Wnd, HMENU Menu) ;

ウィンドウメニューを *Menu* の指定メニューにセットして再描画します。直前のメニューは破棄されません。

引数 *Wnd*

ウィンドウハンドル。

Menu

新しいメニュー，ただし現在のメニューを削除する場合は 0。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***DestroyMenu***

SetMenuItemBitmaps

定義 BOOL SetMenuItemBitmaps (HMENU Menu, WORD Position, WORD Flags,
HBITMAP BitmapUnchecked,
HBITMAP BitmapChecked) ;

2 つのビットマップにメニュー項目を割り当てます。項目チェック時の表示用と，チェック解除時の表示用です。

引数 *Menu*

メニューハンドル。

Position

コマンド ID またはメニュー項目位置。

Flags

MF_BYCOMMAND または ***MF_BYPOSITION*** のいずれか（第 7 章の「**MF_メニューフラグ**」を参照）。

BitmapUnchecked

項目のチェックがないときに表示する ***HBITMAP***。何も表示しない場合は 0。

BitmapChecked

項目のチェック時に表示する ***HBITMAP***。何も表示しない場合は 0。

BitmapUncheked も *BitmapChecked* も 0 の場合は，デフォルトのチェックマークが使用されます。

戻り値　成功すれば 0 以外，失敗すれば 0。

SetMessageQueue

定義　BOOL SetMessageQueue (int Msg) ;

アプリケーションメッセージキューを指定サイズで新たに作成します。直前のキューは削除されます。

引数　*Msg*

キューのサイズ。

戻り値　成功すれば 0 以外，失敗すれば 0。

SetMetaFileBits

定義　HANDLE SetMetaFileBits (HANDLE Mem) ;

Mem の指定に基づきメモリメタファイルを作成します。

引数　*Mem*

GetMetaFileBits で直前に作成されたメタファイルデータを含むグローバルメモリブロック。

戻り値　成功すればメモリメタファイルハンドル，失敗すれば 0。

SetPaletteEntries

定義 WORD SetPaletteEntriess (HPALETTE Palette, WORD StartIndex,
WORD NumEntries,
LPPALETTEENTRY PaletteEntries) ;

論理パレットの指定範囲のエントリを *PaletteEntries* の値にセットします。

引数 *Palette*

論理パレットハンドル。

StartIndex

セットされる最初のエントリ。

NumEntries

セットされるエントリ数。

PaletteEntries

PALETTEENTRY 構造体の配列。

戻り値 実際にセットされたエントリ数，ただしエラーの場合は 0。

SetParent

定義 HWND SetParent (HWND WndChild, HWND WndNewParent) ;

子ウィンドウの親ウィンドウを *WndNewParent* に変更します。

引数 *WndChild*

子ウィンドウハンドル。

WndNewParent

親ウィンドウハンドル。

戻り値 直前の親ウィンドウ。

SetPixel

定義 DWORD SetPixel (HDC DC, int X, int Y, COLORRFF Color) ;

指定座標位置にピクセルを描画します。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

描画位置の論理座標。

Color

ピクセルの描画色を指定する ***COLORREF***。

戻り値 描画に実際使用された ***COLORREF***，ただし描画位置がクリッピング領域外の場合は－1。

SetPolyFillMode

定義 int SetPolyFillMode (HDC DC, int PolyFillMode) ;

多角形塗りつぶしモードを GDI 関数にセットして，多角形アルゴリズムで内部座標位置の計算を行ないます。

引数 *DC*

デバイスコンテキストハンドル。

PolyFillMode

ALTERNATE または ***WINDING*** のいずれか（第 7 章の「**PolyFill モード**」を参照）。

戻り値 成功すれば直前の塗りつぶしモード，失敗すれば 0。

SetProp

定義 BOOL SetProp (HWND Wnd, LPSTR Str, HANDLE Data) ;

Str で指定したウィンドウプロパティリストのエントリの変更，または追加を行ないます。

引数 *Wnd*

ウィンドウハンドル。

Str

ヌルで終わる文字列，または ***AddAtom*** を呼び出して生成したアトム値。

Data

対応するプロパティデータハンドル。

戻り値 追加できれば 0 以外，できなければ 0。

SetRect

定義 void SetRect (LPRECT Rect, int X 1, int Y 1, int X 2, int Y 2) ;

Rect に指定座標を設定します。

引数 *Rect*

格納する ***RECT*** 構造体。

X 1, Y 1

矩形の左上隅。

X 2, Y 2

矩形の右下隅。

SetRectEmpty

定義 void SetRectEmpty (LPRECT Rect) ;

Rect 座標すべてに 0 をセットします。

引数 *Rect*

格納する ***RECT*** 構造体。

SetRectRgn

定義 void SetRectRgn (HRGN Rgn, int X 1, int Y 1, int X 2, int Y 2) ;

Rgn の割り当てスペースに指定サイズの矩形領域を作成します。

引数 *Rgn*

領域ハンドル。

X 1, Y 1

矩形領域の左上隅。

X 2, Y 2

矩形領域の右下隅。

関連項目 ***CreateRectRgn***

S
Fu

SetResourceHandler

定義 FARPROC SetResourceHandler (HANDLE Instance, LPSTR ResType,
FARPROC LoadFunc) ;

リソースのロック（***LockResource***）の際に Windows が呼び出すコールバック関数を設定します。コールバック関数に渡されるのは格納するリソースの *Mem*, *Instance*, ***FindResource*** からの *ResInfo* です。

引数 *Instance*

実行形式ファイルにリソースを含むモジュールインスタンス。

ResType

リソースタイプを指定する short int へのポインタ。

LoadFunc

コールバック関数の手続きインスタンスアドレス。

戻り値 コールバック関数へのポインタ。

SetROP 2

定義 int SetROP 2 (HDC DC, int DrawMode) ;

DrawMode の指定値に現在の描画モードをセットします。オブジェクトインテリアとペンを画面にすでにある色に結合する方法を指定します。

引数 *DC*

デバイスコンテキストハンドル。

DrawMode

R 2_定数のいずれか（第 7 章の「**R 2_バイナリラスタ操作定数**」を参照）。

戻り値 直前の描画モード。

SetScrollPos

定義 int SetScrollPos (HWND Wnd, int Bar, int Pos, BOOL Redraw) ;

スクロールバーのスクロールボックスの位置を *Pos* にセットします。

引数 *Wnd*

ウィンドウハンドル，ただし *Bar* を ***SB_CTL*** に設定した場合はスクロールバーコントロールハンドル。

Bar

SB_CTL, SB_HORZ, SB_VERT のいずれか（第7章の「**SB_スクロールバー定数**」を参照）。

Pos

新しい位置。

Redraw

スクロールバーの再描画には0以外。

戻り値 スクロールボックスの直前の位置。

SetScrollRange

定義 void SetScrollRange (HWND Wnd, int Bar, int MinPos, int Maxpos, BOOL Redraw) ;

スクロールバーの最小位置と最大位置の値をセットします。

引数 *Wnd*

ウィンドウハンドル，ただし *Bar* を ***SB_CTL*** に設定した場合はスクロールバーコントロールハンドル。

Bar

SB_CTL, SB_HORZ, SB_VERT のいずれか（第7章の「**SB_スクロールバー定数**」を参照）。

MinPos

最小スクロール位置。

MaxPos

最大スクロール位置。*MinPos* とともに0にセットした場合はスクロールバーを非表示。

Redraw

スクロールバーの再描画には0以外。

SetSoundNoise

定義 int SetSoundNoise (int Source, int Duration) ;

演奏デバイスのノイズソースと継続時間をセットします。

引数 *Sourse*

次の定数のいずれか（第 7 章の「**S_サウンド定数**」を参照)。

S_PERIOD 512, S_PERIOD 1024, S_PERIOD 2048, S_PERIODVOICE, S_WHITE 512, S_WHITE 1024, S_WHITE 2048, S_WHITEVOICE

Duration

ノイズ継続時間（クロックチック数による)。

戻り値 成功すれば 0, 無効な *Source* の場合は ***S_SERDSR***。

SetStretchBltMode

定義 int SetStretchBltMode (HDC DC, int StretchMode) ;

StretchMode のストレッチングモードをセットして，ビットマップを縮小します。

引数 *DC*

デバイスコンテキストハンドル。

StretchMode

BLACKONWHITE, COLORONCOLOR, WHITEONBLACK のいずれか（第 7 章の「**StretchBlt モード**」を参照)。

戻り値 直前のストレッチングモード。

SetSwapAreaSize

定義 DWORD SetSwapAreaSize (WORD Size) ;

アプリケーションのコードセグメントに使用するメモリ量を，Windows ロード後の残余メモリの半分まで増やします。

引数 *Size*
16 バイトのパラグラフ数。

戻り値 下位ワードで実際に得られたパラグラフ数，上位ワードで最大サイズ。

SetSysColors

定義 void SetSysColors (int Changes, LPINT SysColor,
LONG FAR *ColorValues) ;

SysColor の指定表示要素のシステムカラーをグローバルセットします。

引数 *Change*
変更するシステムカラー数。

SysColor
インデックスが ***COLOR*** _定数である整数型配列（第 7 章の「**COLOR_システムカラーコード**」を参照）。

ColorValues
SysColor のカラーインデックスに対応する RGB カラー値を含む ***DWORD*** 配列。

SetSysModalWindow

定義　HWND SetSysModalWindow (HWND Wnd) ;

Wnd をシステムモードつきのウィンドウにします。システムモード状態はウィンドウの破棄とともに終了します。

引数　*Wnd*
ウィンドウハンドル。

戻り値　直前のシステムモードつきウィンドウ。

SetSystemPaletteUse

定義　WORD SetSystemPaletteUse (HDC DC, WORD Usage) ;

入力フォーカスがあるアプリケーションに対して，システムパレットを全面的に使えるようにします。

引数　*DC*
デバイスコンテキストハンドル。

Usage
SYSPAL_NOSTATIC または ***SYSPAL_STATIC*** のいずれか（第 7 章の「**SYSPAL_システムパレットフラグ**」を参照）。

戻り値　直前のシステムパレット使用法。

関連項目　***GetSysColor, SetSysColors, UnrealizeObject, WM_SYSCOLORCHANGE***

SetTextAlign

定義 WORD SetTextAlign (HDC DC, WORD Flags) ;

TextOut と ***ExtTextOut*** のテキスト配置フラグをセットして，境界矩形との相対位置を決定します。

引数 *DC*

デバイスコンテキストハンドル，またはディスプレイコンテキストハンドル。

Flags

次の定数の組み合わせ（第 7 章の「**TA_テキスト配置オプション**」を参照）。

TA_BASELINE, TA_BOTTOM, TA_CENTER, TA_LEFT, TA_NOUPDATECP, TA_RIGHT, TA_TOP, TA_UPDATECP

戻り値 下位ワードで直前の水平配置，上位ワードで直前の垂直配置。

SetTextCharacterExtra

定義 int SetTextCharacterExtra (HDC DC, int CharExtra) ;

GDI によるテキスト行作成時の文字間隔をセットします。

引数 *DC*

デバイスコンテキストハンドル。

CharExtra

文字間隔（論理単位）。

戻り値 直前の文字間の間隔。

SetTextColor

定義 DWORD SetTextColor (HDC DC, COLORREF Color) ;

TextOut と ***ExtTextOut*** のテキスト色，またはデバイスがサポートする中で最も近い色をセットして字体を描画します。GDI によるビットマップのカラーとモノクロの相互変換にも使用されます。

引数 *DC*

デバイスコンテキストハンドル。

Color

テキストの ***COLORREF***。

戻り値 直前のテキストの RGB カラー値。

関連項目 ***SetBkColor, SetBkMode***

SetTextJustification

定義 int SetTextJustification (HDC DC, int BreakExtra, int BreakCount) ;

GDI が使う位置合わせ引数を定義して，テキスト行の位置合わせを行ないます。

引数 *DC*

デバイスコンテキストハンドル。

BreakExtra

追加行間隔スペース（論理単位)。

BreakCount

行当たりのブレーク文字数（通常，空白文字)。

戻り値 成功すれば 1, 失敗すれば 0。

関連項目 ***GetTextExtent***, ***GetTextMetrics***, ***TextOut***

SetTimer

定義 WORD SetTimer (HWND Wnd, int IDEvent, WORD Elapse, FARPROC TimerFunc) ;

システムタイマーを作成して，***WM_TIMER*** メッセージを *Elapse* の指定間隔でアプリケーションに送り出します。

引数 *Wnd*

ウィンドウハンドル，ただし対応ウィンドウがなければ 0。

IDEvent

0 以外のタイマーイベント ID，ただし *Wnd* が 0 の場合は無視。

Elapse

タイマーイベント間のミリ秒数。

TimerFunc

コールバック関数の手続きインスタンスアドレス，ただしアプリケーションのキューに ***WM_TIMER*** メッセージを置く場合は NULL。

戻り値 *Wnd* が 0 でなければ ***IDEvent***, *Wnd* が 0 であれば新タイマーイベント，エラーであれば 0。

関連項目 ***WM_TIMER***

SetViewportExt

定義 DWORD SetViewportExt (HDC DC, int X, int Y) ;

ビューポートの X 軸方向範囲と Y 軸方向範囲をセットします。論理単位をデバイス単位に合わせるために，GDI によって伸長または圧縮する範囲が定義されます。

引数 *DC*

デバイスコンテキストハンドル。

X, Y
新しいビューポート範囲（デバイス単位)。

戻り値 下位ワードで直前のX軸方向範囲，上位ワードで直前のY軸方向範囲。

SetViewportOrg

定義 DWORD SetViewportOrg (HDC DC, int X, int Y) ;

ビューポート原点をセットします。論理座標からデバイス座標位置へのマップ方法が定義されます。

引数 *DC*
デバイスコンテキストハンドル。

X, Y
新しいビューポート原点（デバイス単位)。

戻り値 下位ワードで直前の原点のX座標，上位ワードでY座標。

SetVoiceAccent

定義 int SetVoiceAccent (int Voice, int Tempo, int Volume, int Mode, int Pitch) ;

音声キューのアクセントを置き換えます。

引数 *Voice*
音声キュー（1以上)。

Tempo
1分当たりに演奏される4分音符の数（デフォルトは120)。

Volume
ボリュームレベル（0から255)。

Mode
S_LEGATO, S_NORMAL, S_STACCATO のいずれかの定数(第7章の「**S_サウンド定数**」を参照)。

Pitch
音符の幅（0から83)。

戻り値 成功すれば0, 失敗すれば次の負の定数のいずれか（第7章の「**S_サウンド定数**」を参照)。

S_SERDMD, S_SERDTP, S_SERDVL, S_SERQFUL

SetVoiceEnvelope

定義 int SetVoiceEnvelope (int Voice, int Shape, int RepeatCount) ;

音声キューの音声エンベロープを置き換えます。

引数 *Voice*
音声キュー。

Shape
OEM 波形テーブルインデックス。

RepeatCount
音当たりの波形の繰り返し数。

戻り値 成功すれば0, 失敗すれば ***S_SERQFUL*** または ***S_SERDSH*** のいずれかの負の定数（第7章の「**S_サウンド定数**」を参照)。

SetVoiceNote

定義 int SetVoiceNote (int Voice, int Value, int Length, int Cdots) ;

指定の音声キューに音符を設定します。

引数 *Voice*

音声キュー。

Value

1から84までの音程，休符は0。

Length

音符持続時間の逆数。

Cdots

付点による音符持続時間（*Length*＊（*Cdots*＊3/2)）。

戻り値 成功すれば0, 失敗すれば次の負の定数のいずれか（第7章の「**S_サウンド定数**」を参照)。

S_SERDCC, S_SERDLN, S_SERBDNT, S_SERQFUL

SetVoiceQueueSize

定義 int SetVoiceQueueSize (int Voice, int Bytes) ;

演奏中でない音声キューのサイズをセットします。デフォルトは192バイトで，約32個の音符に相当します。

引数 *Voice*

音声キュー。

Bytes

音声キューのサイズ（バイト数)。

戻り値 成功すれば 0, 失敗すれば ***S_SERMACT*** または ***S_SEROFM*** のいずれかの負の定数（第 7 章の「**S_サウンド定数**」を参照)。

SetVoiceSound

定義 int SetVoiceSound (DWORD Voice, DWORD Frequency, int Duration) ;

指定した音声周波数と持続時間を音声キューに入れます。

引数 *Voice*

音声キュー。

Frequency

上位ワードで周波数の整数部(Hz), 下位ワードで小数部。

Duration

音声持続時間（クロック単位)。

戻り値 成功すれば 0, 失敗すれば ***S_SERDDR, S_SERDFQ, S_SERDVL, S_SERQFUL*** のいずれかの定数（第 7 章の「**S_サウンド定数**」を参照)。

SetVoiceThreshold

定義 int SetVoiceThreshold (int Voice, int Notes) ;

音声キューの限界値レベルをセットします。キュー内のノート数が限界値を下回れば，限界値フラグがセットされます。

引数 *Voice*

音声キュー。

Notes

限界値レベルを指定する音符数。

戻り値 成功すれば 0, 音符数が範囲外であれば 1。

関連項目 *GetThresholdStatus*

SetWindowExt

定義 DWORD SetWindowExt (HDC DC, int X, int Y) ;

ウィンドウのX軸方向範囲とY軸方向範囲をセットします。ビューポート範囲と同様に，デバイス単位に合わせて論理単位を伸長または圧縮する範囲を定義します。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

ウィンドウ範囲。

戻り値 下位ワードで直前のX軸方向範囲，上位ワードで直前のY軸方向範囲，ただしエラーの場合は0。

SetWindowLong

定義 DWORD SetWindowLong (HWND Wnd, int Index, DWORD NewLong) ;

ウィンドウクラス構造体の属性を新しい値にします。

引数 *Wnd*

ウィンドウハンドル。

Index

GWL_EXSTYLE, GWL_STYLE, GWL_WNDPROC，または後に割り当てる4バイト値にアクセスするための正のバイトオフセット（第7章の「**GWL_ウィンドウフィールドオフセット**」を参照）。

NewLong

置き換え値。

戻り値　直前の値。

SetWindowOrg

定義　DWORD SetWindowOrg (HDC DC, int X, int Y) ;

指定デバイスコンテキストのビューポート内に，ウィンドウの原点をセットします。

引数　*DC*

デバイスコンテキストハンドル。

X, Y

新しいウィンドウ原点。

戻り値　下位ワードで直前の X 座標，上位ワードで直前の Y 座標。

SetWindowPos

定義　void SetWindowPos (HWND Wnd, HWND WndInsertAfter, int X, int Y, int cx, int cy, WORD Flags) ;

ウィンドウのサイズ，位置，順序を変更します。

引数　*Wnd*

ウィンドウハンドル。

WndInsertAfter

ウィンドウマネージャリストの中で1つ前にあるウィンドウ。

X, Y

左上隅。

cx

新しいウィンドウの幅。

cy
新しいウィンドウの高さ。

Flags
次のいずれかの定数（第 7 章の「**SWP_ウィンドウ位置設定フラグ**」を参照)。

SWP_DRAWFRAME, SWP_HIDEWINDOW, SWP_NOACTIVATE, SWP_NOREMOVE, SWP_NOSIZE, SWP_NOREDRAW, SWP_NOZORDER, SWP_SHOWWINDOW

SetWindowsHook

定義 FARPROC SetWindowsHook (int FilterType, FARPROC FilterFunc) ;

FilterType の指定フィルタ関数のチェインに，フィルタ関数を 1 つインストールします。この関数に渡されるのは，値がフィルタタイプに依存する *Code*, *wParam*, *lParam* です。

引数 *FilterType*
次の定数のいずれか（第 7 章の「**WH_Windows フックコード**」を参照)。

WH_CALLWNDPROC, WH_GETMESSAGE, WH_JOURNALPLAYBACK, WH_JOURNALRECORD, WH_KEYBOARD, WH_MSGFILTER, WH_SYSMSGFILTER

FilterFunc
フィルタ関数の手続きインスタンスアドレス。

戻り値 旧導入フィルタ関数の手続きインスタンスアドレス，直前のフィルタがない場合は NULL。

関連項目 ***DefHookProc***

SetWindowText

定義 void SetWindowText (HWND Wnd, LPSTR Str) ;

ウィンドウのタイトルタイトルまたはコントロールのテキストを，*Str* の指定文字列にします。

引数 *Wnd*

ウィンドウまたはコントロールハンドル。

Str

ヌルで終わる文字列。

SetWindowWord

定義 WORD SetWindowWord (HWND Wnd, int Index, WORD NewWord) ;

ウィンドウクラス構造体の *Index* 位置にある属性値を変更します。

引数 *Wnd*

ウィンドウハンドル。

Index

GWW_APPLICATION -> ***HINSTANCE***, ***GWW_HWNDPARENT***, ***GWW_ID***, または後に割り当てる2バイト値にアクセスするための正のバイトオフセット（第7章の「**GWW_ウィンドウフィールドオフセット**」を参照)。

NewWord

置き換え値。

戻り値 直前の値。

ShowCaret

定義 void ShowCaret (HWND Wnd) ;

画面上の *Wnd* が所有するキャレットを表示します。

引数 *Wnd*

ウィンドウハンドル，ただし現在のタスク内のウィンドウの場合は 0。

ShowCursor

定義 int ShowCursor (BOOL Show) ;

カーソルの表示カウント（初期セット値 0）が 0 のままか 0 以上になると，カーソルを表示します。それ以外は非表示になります。

引数 *Show*

表示カウントをインクリメントする場合は 0 以外，デクリメントする場合は 0。

戻り値 新しい表示カウント。

ShowOwnedPopups

定義 void ShowOwnedPopups (HWND Wnd, BOOL Show) ;

特定ウィンドウの対応ポップアップウィンドウすべてを，*Show* の指定どおりに表示または非表示にします。

引数 *Wnd*

ウィンドウハンドル。

Show
非表示のポップアップウィンドウをすべて表示する場合は 0 以外，表示ポップアップウィンドウをすべてを非表示にする場合は 0。

ShowScrollBar

定義 void ShowScrollBar (HWND Wnd, WORD Bar, BOOL Show) ;

スクロールバーを *Show* の指定どおりに表示または非表示にします。

引数 *Wnd*
ウィンドウハンドル，ただし *Bar* を ***SB_CTL*** にセットした場合はスクロールバーコントロール。

Bar
SB_BOTH, SB_CTL, SB_HORZ, SB_VERT のいずれかの定数（第 7 章の「**SB_スクロールバー定数**」を参照）。

Show
スクロールバーを表示する場合は 0 以外，スクロールバーを非表示にする場合は 0。

ShowWindow

定義 BOOL ShowWindow (HWND Wnd, int CmdShow) ;

ウィンドウを *CmdShow* の指定どおりに表示または非表示にします。

引数 *Wnd*
ウィンドウハンドル。

CmdShow
SW_定数のいずれか（第 7 章の「**SW_ウィンドウ表示定数**」を参照）。

戻り値 ウィンドウが表示されていた場合は 0 以外，非表示であった場合は 0。

SizeofResource

定義 WORD SizeofResource (HANDLE Instance, HANDLE ResInfo) ;

リソースのサイズを取得します。取得サイズは，境界合わせの関係で実際より大きい場合があります。

引数 *Instance*

実行形式ファイルにリソースを含むモジュールインスタンス。

ResInfo

FindResource によって返される要求されたリソース。

戻り値 リソースのサイズ（バイト数），該当リソースがなければ 0。

関連項目 ***AccessResource***

StartSound

定義 int StartSound () ;

音声キューすべてを消去せずに演奏します。

戻り値 ありません。

StopSound

定義 int StopSound () ;

音声キューすべての演奏を中止して，キュー内容をフラッシュし，さらに音声サウンドドライバを終了します。

戻り値 ありません。

StretchBlt

定義 BOOL StretchBlt (HDC DestDC, int X, int Y, int Width, int Height,HDC SrcDC, int XSrc, int YSrc, int SrcWidth, int SrcHeight, DWORD Rop) ;

ビットマップを伸長または圧縮して，転送元の矩形から転送先の矩形に移動します。転送元と転送先の矩形の組み合わせ方法は *Rop* によって指定されます。

引数 *DestDC*

対象デバイスコンテキストのハンドル。

X, Y

対象矩形の左上隅。

Width

対象矩形の幅（論理単位)。

Height

対象矩形の高さ（論理単位)。

SrcDC

転送元のビットマップデバイスコンテキスト。

XSrc, YSrc

転送元の矩形の左上隅。

SrcWidth

転送元の矩形の幅（論理単位)。

SrcHeight

転送元の矩形の高さ（論理単位)。

Rop

実行されるラスタ操作（第7章の「**Ternary Raster Operation（3次元ラスタ操作)**」を参照)。

戻り値 ビットマップが描ければ0以外，描けなければ0。

関連項目 *SetStretchBltMode*

StretchDIBits

定義 int StretchDIBits (HDC DC, WORD DestX, WORD DestY, WORD DestWidth, WORD DestHeight, WORD SrcX, WORD SrcWidth, WORD SrcHeight,LPSTR Bits, LPBITMAPINFO BitsInfo, WORD Usage, DWORD Rop) ;

デバイス独立のビットマップである DIB を伸長または圧縮して，転送元の矩形から転送先の矩形に移動します。転送元と転送先の矩形の組み合わせ方法は *Rop* によって指定されます。

引数 *DC*

格納するデバイスコンテキストのハンドル。

DestX, DestY

転送先の矩形原点（論理単位)。

DestWidth

転送先の矩形の X 軸方向範囲（論理単位)。

DestHeight

転送先の矩形の Y 軸方向範囲（論理単位)。

SrcX, SrcY

DIB の転送元矩形原点（ピクセル単位)。

SrcWidth

転送元矩形の幅（ピクセル単位)。

SrcHeight

転送元矩形の高さ（ピクセル単位)。

Bits

DIB ビットを含む ***BYTE*** 配列。

Usage

DIB_RGB_COLOR は *BitsInfo* を指定，*bmiColors* フィールドは RGB 値を含有，***DIB_PAL_COLORS*** は現在実現されている論理パレットにインデックスを指定（第 7 章の「**DIB_カラーテーブル識別子**」を参照)。

Rop

3 次元ラスタ操作定数のいずれか（第 7 章の「**Ternary Raster Operation（3 次元ラスタ操作）**」を参照）。

戻り値　コピーされた走査ライン数。

SwapMouseButton

定義　BOOL SwapMouseButton (BOOL Swap) ;

マウスの左右ボタンの機能を，*Swap* の指定どおりに切り換えまたは復元します。

引数　*Swap*

ボタン機能の切り換えには 0 以外，前の機能を復元するには 0。

戻り値　機能が切り換えられれば 0 以外，切り換えられなければ 0。

SwapRecording

定義　void SwapRecording (WORD Flag) ;

Windows の ***Swap*** プログラム実行時のスワップ動作分析を開始または終了します。

引数　*Flag*

0 は分析中止，1 はスワップの呼び出し，復帰，廃棄の記録，2 は 1 の内容およびサンク経由の呼び出しの記録。

SwitchStackBack

定義 void SwitchStackBack () ;

現在のタスクのスタックをデータセグメントに復元します。AX : DX レジスタの内容はそのまま保存されます。

関連項目 ***SwitchStackTo***

SwitchStackTo

定義 void SwitchStackTo (WORD StackSegment, WORD StackPointer, WORD StackTop) ;

現タスクのスタックを *StackSegment* に変更します。DS＝SS を仮定している DLL 関数の場合には，そのスタックをデータセグメントにセットすることもできます。

引数 *StackSegment*
スタックを含むデータセグメント。

StackPointer
StackSegment への先頭スタックオフセット。

StackTop
StackPointer からのスタックオフセットの先頭。

関連項目 ***SwitchStackBack***

SyncAllVoices

定義 int SyncAllVoices () ;

音声キューすべてに同期マークを入れます。

戻り値 成功すれば 0, 音声キューがいっぱいの場合は ***S_SERQFUL***（第 7 章の「**S_サウンド定数**」を参照）。

TabbedTextOut

定義 DWORD TabbedTextOut (HDC DC, int X, int Y, LPSTR Str, int Count,
int TabPositions, LPINT TabStopPositions,
int TabOrigin) ;

タブを *TabStopPositions* の指定幅に展開して，選択フォントでテキスト行を描きます。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

文字列開始点。

Str

描かれる文字列。

Count

Str の文字数。

TabPositions

TabStopPositions のタブストップの数，ただしタブストップの展開幅が平均 8 文字の場合は 0。

TabStopPositions

昇順のタブストップ位置を含む整数型配列（ピクセル単位)。

TabOrigin

タブ展開の開始位置（論理単位)。

戻り値 ありません。

TextOut

定義 BOOL TextOut (HDC DC, int X, int Y, LPSTR Str, int Count) ;

現在の選択フォントでテキスト行を描きます。

引数 *DC*

デバイスコンテキストハンドル。

X, Y

文字列開始点。

Str

描かれる文字列。

Count

Str の文字数。

戻り値 描ければ 0 以外，描けなければ 0。

関連項目 ***SetTextAlign***

Throw

定義 void Throw (LPCATCHBUF CatchBuf, int ThrowBack) ;

アプリケーションの実行環境を復元します。***Catch*** 関数で *CatchBuf* に元の環境を保存した時点での処理が継続されます。

引数 *CatchBuf*

実行環境を含む ***CATCHBUF*** 構造体。

ThrowBack

Catch 関数が返した値。

関連項目 ***Catch***

ToAscii

定義 int ToAscii (WORD VirtKey, WORD ScanCode, LPSTR KeyState, LPVOID CharBuff, WORD Flags) ;

VirtKey と現在のキーボード状態を変換して，対応する ANSI 文字にします。

引数 *VirtKey*

仮想キーコード。

ScanCode

未処理のキー走査コード。キーが離された場合は上位ビットをセット。

KeyState

各キーの状態を含む 256 バイトの配列。キーが離された場合は上位ビットをセット。

CharBuff

32 ビット格納バッファへのポインタ。

Flags

使いません。

戻り値 2 であればアクセントとデッドキーが *CharBuff* にコピー，1 であれば ANSI 文字が 1 つ *CharBuff* にコピー，0 であれば現在のキーボード状態では変換不能。

TrackPopupMenu

定義 BOOL TrackPopupMenu (HMENU Menu, WORD Flags, int x, int y, int cx,
HWND Wnd, LPRECT Rect) ;

フローティングポップアップメニューを表示して，選択項目をトラックします。フローティングポップアップメニューの画面表示位置は不定です。

引数 *Menu*

ポップアップメニューハンドル。

Flags

使わないので 0 にセット。

x, y

メニューの左上隅（画面座標系）。

cx

メニュー幅（画面単位），ただしデフォルトの場合は 0。

Wnd

ポップアップメニューのオーナーウィンドウ（***WM_COMMAND*** メッセージの送り先）。

Rect

マウスボタンを離してもメニューが消えないマウス領域を定義する ***RECT*** 構造体。

戻り値 成功すれば 0 以外，失敗すれば 0。

関連項目 ***CreatePopupMenu***, ***GetSubMenu***

TranslateAccelerator

定義　int TranslateAccelerator (HWND Wnd, HANDLE AccTable, LPMSG Msg) ;

キーボードの ***WM_KEYUP*** と ***WM_KEYDOWN*** の2つのアクセラレータを，メニューコマンドメッセージである ***WM_COMMAND*** と ***WM_SYSCOMMAND*** に変換します。変換後はウィンドウに直送されます。

引数　*Wnd*

ウィンドウハンドル。

AccTable

LoadAccelerators によって返されるアクセラレータテーブルハンドル。

Msg

GetMessage または ***PeekMessage*** から取得される ***MSG*** 情報。

戻り値　変換できれば0以外，変換できなければ0。

TranslateMDISysAccel

定義　BOOL TranslateMDISysAccel (HWND Wnd, LPMSG Msg) ;

キーボードアクセラレータをMDI子ウィンドウのシステムメニュー ***WM_SYSCOMMAND*** メッセージに変換してから，ウィンドウに直送します。

引数　*Wnd*

MDIクライアント親ウィンドウ。

Msg

GetMessage または ***PeekMessage*** から取得される ***MSG*** 情報。

戻り値　変換できれば0以外，変換できなければ0。

TranslateMessage

定義　BOOL TranslateMessage (LPMSG Msg) ;

WM_KEYDOWN/UP の組み合わせを変換して，***WM_CHAR*** または ***WM_DEADCHAR*** にします。さらに ***WM_SYSKEYDOWN/UP*** の組み合わせを変換して，***WM_SYSCHAR*** または ***WM_SYSDEADCHAR*** にします。さらに文字メッセージをアプリケーションキューに送ります。

引数　*Msg*

GetMessage または ***PeekMessage*** から取得される ***MSG*** 情報。

戻り値　変換できれば 0 以外，変換できなければ 0。

TransmitCommChar

定義　int TransmitCommChar (int Cid, char AChar) ;

AChar を通信デバイスの送信キューの先頭に置き，ただちに送信します。

引数　*Cid*

通信デバイス。

AChar

送信文字。

戻り値　成功すれば 0, 直前の文字が未送信のために送信できない場合は負の値。

関連項目　***OpenComm***

UngetCommChar

定義 int UngetCommChar (int Cid, char AChar) ;

通信デバイスの受信キューに *AChar* を戻します。

引数 *Cid*

通信デバイス。

AChar

取得解除する文字。

戻り値 成功すれば 0, 失敗すれば負の値。

関連項目 ***OpenComm***

UnhookWindowsHook

定義 BOOL UnhookWindowsHook (int Hook, FARPROC HookFunc) ;

Hook で指定したフック関数チェインからフック関数を 1 つ除去します。

引数 *Hook*

次のいずれかの定数（第 7 章の「**WH_Windows フックコード**」を参照)。

WH_CALLWNDPROC, WH_GETMESSAGE,
WH_JOURNALPLAYBACK, WH_JOURNALRECORD,
WH_KEYBOARD, WH_MSGFILTER

HookFunc

フック関数の手続きインスタンスアドレス。

戻り値 成功すれば 0 以外, 失敗すれば 0。

U
Fun

UnionRect

定義 int UnionRect (LPRECT DestRect, LPRECT SrclRect, LPRECT, Src 2 Rect) ;

2つの矩形の共用体を作成後，*DestRect* に格納します。

引数 *DestRect*

共用体の ***RECT*** 構造体。

Src 1 Rect

ソースの ***RECT*** 構造体 1。

Src 2 Rect

ソースの ***RECT*** 構造体 2。

戻り値 共用体が空でなければ 0 以外，空であれば 0。

UnlockData

定義 HANDLE UnlockData (int Dummy) ;

移動可能な現在のデータセグメントのロックを解除するマクロです。

引数 *Dummy*

使わないので 0 にセット。

戻り値 ロックを解除されたセグメントのハンドル，失敗すれば 0。

UnlockResource

定義 BOOL UnlockResource (HANDLE ResData) ;

ResData リソースのロックを解除して，参照カウントをデクリメントするマクロです。

引数 *RezData*

グローバルメモリブロックハンドル。

戻り値 参照カウントが 0 であれば 0, 参照カウントが 0 でなければ 0 以外。

UnlockSegment

定義 HANDLE UnlockSegment (WORD Segment) ;

Segment で指定したセグメントのロックを解除します。

引数 *Segment*

セグメントアドレス。現在のデータセグメントのロックを解除するには - 1。

戻り値 参照カウントが 0 にデクリメントされれば 0, されなければ 0 以外。

関連項目 ***LockSegment***

UnrealizeObject

定義 BOOL UnrealizeObject (HBRUSH HObject) ;

HObject がブラシであれば，次回選択時の原点を GDI によってリセットします。*HObject* が論理パレットであれば，パレットを実現します。

引数 *HObject*

リセットされるオブジェクト。

戻り値 成功すれば 0 以外，失敗すれば 0。

UnregisterClass

定義 BOOL UnregisterClass (LPSTR ClassName, HANDLE Instance) ;

ウィンドウクラステーブルからウィンドウクラスを削除して，対応するメモリをすべて解放します。

引数 *ClassName*

以前に登録したクラス名（ヌルで終わる文字列)。

Instance

クラスを作成するモジュールインスタンス。

戻り値 成功すれば 0 以外，無効 *ClassName* または該当クラスのウィンドウが終了すれば 0。

関連項目 ***RegisterClass***

UpdateColors

定義 int UpdateColors (HDC DC) ;

クライアント領域の現在のカラーをピクセル単位でシステムパレットに合わせて，クライアント領域を更新します。

引数 *DC*

デバイスコンテキストハンドル。

戻り値 ありません。

UpdateWindow

定義 void UpdateWindow (HWND Wnd) ;

ウィンドウの更新領域が空でなければ，ウィンドウ関数に直接 ***WM_PAINT*** メッセージを送ります。

引数 *Wnd*

ウィンドウハンドル。

ValidateCodeSegments

定義 void ValidateCodeSegments () ;

メモリに対するランダムな書き込みによるコードセグメントの変更があれば，デバッグ情報を端末に出力します。Windows のデバッグバージョン以外では使用できません。関数を無効にするには，WIN.INI の **kernel** セクションの *EnableSegmentChecksum* フラグを 0 にセットします。

この関数は Windows のスタンダードモードと 386 エンハンスモードでは使用できません。

ValidateFreeSpaces

定義 LPSTR ValidateFreeSpaces () ;

空きメモリセグメントをすべてチェックし，内容が正しいかを確認します。この関数が有効に働くのは Windows のデバッグバージョンに限られます。

戻り値 ありません。

ValidateRect

定義　void ValidateRect (HWND Wnd, LPRECT Rect) ;

ウィンドウの更新領域から *Rect* を除去して，クライアント領域を有効にします。

引数　*Wnd*

ウィンドウハンドル。

Rect

更新領域から除去される ***RECT*** 構造体（クライアント座標），ただしクライアント領域全体の場合は NULL。

関連項目　***BeginPaint***

ValidateRgn

定義　void ValidateRgn (HWND Wnd, HRGN Rgn) ;

ウィンドウの更新領域から *Rgn* の指定領域を除去して，クライアント領域を有効にします。

引数　*Wnd*

ウィンドウハンドル。

Rgn

領域ハンドル（クライアント座標）。

VkKeyScan

定義　WORD VkKeyScan (WORD AChar) ;

AChar を変換して，対応する仮想キーコードとシフト状態にします。

引数 *AChar*

対応する仮想キーコードに変換する ANSI 文字。

戻り値 下位バイトで仮想キーコード，上位バイトで下記のシフト状態。

0 はシフトなし，1 は ***SHIFT***，2 は ***CTRL***，6 は ***CTRL-GRPH***，7 は ***SHIFT-CTRL-GRPH***，3 と 4 と 5 は文字に対しては不使用。エラーの場合は上下位バイトとも - 1。

WaitMessage

定義 void WaitMessage () ;

他のアプリケーションに制御権を譲り，アプリケーションキューのメッセージが有効になれば制御が戻ります。

WaitSoundState

定義 int WaitSoundState (int State) ;

演奏ドライバが *State* の指定状態になるのを待ちます。

引数 *State*

S_ALLTHRESHOLD, S_QUEUEEMPTY, S_THRESHOLD のいずれか (第 7 章の「**S_サウンド定数**」を参照)。

戻り値 成功すれば 0, 無効な *State* の場合は ***S_SERDST***。

W
Fun

WindowFromPoint

定義 HWND WindowFromPoint (POINT Point) ;

指定座標位置を含むウィンドウを判定します。

引数 *Point*

チェック対象の ***POINT***（画面座標）。

戻り値 ウィンドウハンドル，ただし該当ウィンドウが指定座標位置にない場合は 0。

WinExec

定義 WORD WinExec (LPSTR CmdLine, WORD CmdShow) ;

CmdLine の指定アプリケーションを実行します。

引数 *CmdLine*

アプリケーションを起動するコマンドライン（ヌルで終る文字列）。

CmdShow

アプリケーションウィンドウの初期表示形態を指定（***ShowWindow*** を参照）。

戻り値 成功すれば 32 より大きい値，失敗すれば次のいずれか。

0 はメモリ不足，5 はタスクにダイナミックリンクの試行，6 はライブラリに多重データセグメントあり，10 は Windows バージョン不正，11 は無効 EXE ファイル，12 は OS/2 アプリケーション，13 は DOS 4.0 アプリケーション，14 は EXE タイプ不明，15 はプロテクトモードで Windows の以前のバージョンに対して作成された．EXE ロードしようとした場合，16 は MULTIPLE 属性（各インスタンスに 1 つのセグメント）と READWRITE 属性（書き込み可能）を持つ.EXE の 2 番目のインスタンスをロードしようとした場合，17 はラージフレーム EMS モードですでに使っている共有不可能な DLL にリンクするアプリケーションの 2 番目のインスタンスをロードしようとした場合，18 はリアルモードでプロテクトモード専用としてマークされたアプリケーションをロードしようとした場合。

WinHelp

定義 BOOL WinHelp (HWND Wnd, LPSTR HelpFile, WORD Command,
DWORD Data) ;

Command の指定コマンドで Windows のヘルプエンジンを呼び出します。

引数 *Wnd*

ウィンドウハンドル。

HelpFile

パスなどを含むヘルプファイル名（ヌルで終わる文字列)。

Command

次のいずれかの定数（第 7 章の「**HELP_ヘルプコマンド**」を参照)。

HELP_CONTEXT, HELP_HELPONHELP, HELP_INDEX, HELP_KEY, HELP_QUIT, HELP_SETINDEX

Data

Command に ***HELP_CONTEXT*** をセットした場合はコンテキストハンドル番号，*Command* に ***HELP_KEY*** をセットした場合はヘルプトピックキーワード（ヌルで終わる文字列)。

戻り値 成功すれば 0 以外，失敗すれば 0。

WriteComm

定義 int WriteComm (int Cid, LPSTR Buf, int Size) ;

Buf の指定バッファを通信デバイスに出力します。

引数 *Cid*

通信デバイス。

Buf

書き込む文字を含むバッファ。

Size

出力文字数。

戻り値 実際に書き込まれた文字数，エラーの場合はエラー発生前の書き込み文字数を絶対値とする負の値。

WritePrivateProfileString

定義
```
BOOL WritePrivateProfileString (LPSTR ApplicationName, LPSTR
                                KeyName, LPSTR Str,
                                LPSTR FileName) ;
```

FileName で指定ヘッダとキー名を検索し，*Str* の値で置き換えます。

引数 *ApplicationName*

アプリケーションヘッダ名。

KeyName

アプリケーションヘッダ名の下のキー名。セクション全体を削除する場合はNULL。

Str

キーの新名称，ただしキー名を削除する場合は NULL。

FileName

初期化ファイル名（ヌルで終わる文字列)。

戻り値 成功すれば 0 以外，失敗すれば 0。

WriteProfileString

定義 BOOL WriteProfileString (LPSTR ApplicationName, LPSTR KeyName, LPSTR Str) ;

WIN.INI で指定ヘッダとキー名を検索し，*Str* の値で置き換えます。

引数 *ApplicationName*

アプリケーションヘッダ。

KeyName

アプリケーションヘッダ名の下のキー名。アプリケーションのセクション全体を削除する場合は NULL。

Str

キーの新名称値，ただしキー名を削除するだけの場合は NULL。

戻り値 成功すれば 0 以外，失敗すれば 0。

wvsprintf

定義 int wvsprintf (LPSTR Output, LPSTR Format, LPSTR ArgList) ;

文字列を書式化してバッファに格納します。

引数 *Output*

書式化文字を格納するバッファ。

Format

書式制御文字列。

ArgList

書式制御文字列の引数配列。

戻り値 成功すれば *Output* の 0 以外の文字数，失敗すれば *Format* の指定よりも短い値。

Yield

定義　BOOL Yield () ;

現在のタスクを終えて，待ちタスクを開始します。

戻り値　ありません。

第5章

Windows メッセージ

この章では，Windows アプリケーションに送られる Windows メッセージをアルファベット順に記載しています。見出しの下で，メッセージを構成する引数 *lParam* と *wParam* の内容を説明します。また，Windows で定義済みのデフォルト応答と予想される戻り値も解説しています。

引数 *lParam* と *wParam* は ***LONG*** 値と ***WORD*** 値です。それぞれの値で Windows が送る情報は 1 つだけとは限りません。各引数の上位部分と下位部分を別々に使うときは，マクロ ***HIBYTE, LOBYTE, HIWORD, LOWORD*** を使います。

LONG 引数と WORD 引数に加え，メッセージの識別子とメッセージの対象となったウィンドウの識別子も送られます。

BM_GETCHECK

ラジオボタンまたはチェックボックスに対するチェックの有無を判定します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ラジオボタンまたはチェックボックスにチェックがあれば 0 以外，チェックがなければ 0。プッシュボタンの場合は常に 0。

BM_GETSTATE

マウスボタンまたは**スペース**を押したときの，ボタンコントロールの状態を判定します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ハイライト表示のプッシュボタン，入力フォーカスがあるボタン，マウスボタンまたは**スペース**の押下，ボタン上にカーソルがある時のマウスボタンの押下のそれぞれに対しては 0 以外。その他の場合は 0。

BM_SETCHECK

ラジオボタンまたはチェックボックスのチェックマークをつけたり，除去したりします。

引数 *wParam*
2 状態ボタンまたはチェックボックスで *wParam* が 0 の場合，チェックマークがあれば除去，0 以外のときは付加します。3 状態ボタンの場合は，*wParam* が 0 の場合，チェックマークまたは淡色表示があればともに除去します。1 のときはチェックマークを付加します。2 のときはボタンを淡色表示にします。

lParam
ありません。

戻り値 ありません。

BM_SETSTATE

ボタンまたはチェックボックスの状態を変更します。

引数 *wParam*

0 の場合，ボタンまたはチェックボックスは通常どおりに描画されます。0 以外の場合，ボタンがハイライト表示になります。

lParam

ありません。

戻り値 ありません。

BM_SETSTYLE

ボタンのスタイルを変更します。

引数 *wParam*

変更後のボタンスタイルです。詳細については第 7 章『**Windows 定数**』の「**BS_ボタンスタイル**」を参照してください。

lParam

0 のときはボタンの再描画は行なわれません。0 以外で変更後のスタイルが現在のスタイルと異なるときは，ボタンが再描画されます。

戻り値 ありません。

CB_ADDSTRING

コンボボックスのリストボックスに文字列を加えます。

引数 *wParam*

ありません。

lParam

追加文字列（ヌルで終わる文字列）へのポインタ。

戻り値 成功すれば追加文字列のインデックス。文字列を格納する充分なメモリがなければ ***CB_ERRSPACE***，エラーがあれば ***CB_ERR***。

コメント ソートされないリストボックスの場合，文字列の追加位置はリストの末尾になります。***CBS_OWNERDRAWFIXED*** か ***CBS_OWNERDRAWVARIABLE*** があり ***CBS_HASSTRINGS*** のないコンボボックスの場合は，*lParam* が 32 ビット値となります。この値が文字列のかわりに格納されます。コンボボックスのオーナーに ***WM_COMPAREITEM*** メッセージが送られ，追加項目と他項目との比較が行なわれます。

CB_DELETESTRING

コンボボックスのリストボックスから文字列を削除します。

引数 *wParam*

削除されるリストボックス項目のインデックス。

lParam

ありません。

戻り値 *wParam* が有効なインデックスであればリスト内の残存項目数，その他の場合は ***CB_ERR***。

コメント ***CBS_OWNERDRAWFIXED*** か ***CBS_OWNERDRAWVARIABLE*** があり ***CBS_HASSTRINGS*** のないコンボボックスの場合，対応する 32 ビット値が削除されます。***WM_DELETEITEM*** メッセージがコンボボックスのオーナーに送られます。

CB_DIR

ファイル仕様とDOSファイル属性に一致するファイル名を，カレントディレクトリからコンボボックスのリストボックスに加えます。

引数 *wParam*

DOSファイル属性。

lParam

ファイル仕様文字列（ヌルで終わる文字列）へのポインタ。

戻り値 成功すればファイル名追加後のリストの末尾項目のインデックス，メモリ不足で項目が格納できなかった場合は ***CB_ERRSPACE***，エラーの場合は ***CB_ERR***。

CB_FINDSTRING

プレフィクス文字列に最初に一致するリストボックス項目を，コンボボックスから検索します。

引数 *wParam*

検索開始位置のインデックスです。***wParam*** の次の項目から検索が開始されます。検索がリストの末尾までおよぶと，続けて項目0から検索が行なわれます。***wParam*** に戻れば終了します。***wParam*** が-1の場合はリスト全体が項目0から検索されます。

lParam

プレフィクス文字列（ヌルで終わる文字列）へのポインタ。

戻り値 成功すれば最初に一致する項目のインデックス，失敗すれば ***CB_ERR***。

コメント ***CBS_OWNERDRAWFIXED*** か ***CBS_OWNERDRAWVARIABLE*** があり ***CBS_HASSTRINGS*** のないコンボボックスの場合，***lParam*** は32ビット値となり，リスト上の対応する32ビット値と比較されます。

CB_GETCOUNT

コンボボックスのリストボックス内の項目数を返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 リストボックスの項目数。

CB_GETCURSEL

コンボボックスのリストボックス内で，現在選択されている項目のインデックスを返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 選択されている項目がなければ ***CB_ERR***，あれば現在選択されている項目のインデックス。

CB_GETEDITSEL

コンボボックスのエディットコントロール内で選択されたテキストの，最初と最後のインデックスを返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 エディットコントロールのないコンボボックスの場合は ***CB_ERR***，コントロールがある場合は，戻り値の下位ワードは開始インデックス，上位ワードは終了インデックス。

CB_GETITEMDATA

コンボボックスのリストボックス内の項目に対応する 32 ビット値を返します。

引数 *wParam*

項目のインデックス。

lParam

ありません。

戻り値 成功すれば対応する 32 ビットのインデックス，失敗すれば ***CB_ERR***。

CB_GETLBTEXT

コンボボックスのリストボックスから受け取るバッファに項目をコピーします。

引数 *wParam*

項目のインデックス。

lParam

バッファへのポインタ。バッファ容量として，項目の長さと末尾のヌル文字を格納するだけのサイズが必要です。

戻り値 ありません。

コメント ***CBS_OWNERDRAWFIXED*** か ***CBS_OWNERDRAWVARIABLE*** があり ***CBS_HASSTRINGS*** のないコンボボックスの場合，リスト項目とに対応する 32 ビット値がバッファにコピーされます。

CB_GETLBTEXTLEN

コンボボックスのリストボックス項目のバイト長を返します。

引数 *wParam*

項目のインデックス。

lParam

ありません。

戻り値 *wParam* が有効な場合は，そのインデックス位置にある項目の長さ，無効な場合は ***CB_ERR***。

CB_INSERTSTRING

文字列をソートせずに，コンボボックスのリストボックスに挿入します。

引数 *wParam*

－1であれば文字列はリスト末尾に加えられます。－1以外の値は文字列挿入位置のインデックスになります。

lParam

挿入文字列（ヌルで終わる文字列）へのポインタ。

戻り値 成功すれば文字列挿入位置のインデックス，メモリ不足で文字列を格納できなかった場合は ***CB_ERRSPACE***，エラーの場合は ***CB_ERR***。

CB_LIMITTEXT

コンボボックスのエディットコントロールへの入力文字数を制限します。

引数 *wParam*

新しく設定する最大文字数。0であれば文字数に制限はありません。

lParam

ありません。

戻り値 成功すれば0以外，失敗すれば0。エディットコントロールのないコンボボックスの場合には ***CB_ERR***。

CB_RESETCONTENT

コンボボックスのリストボックスから項目をすべて除去します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント ***CBS_OWNERDRAWFIXED*** か ***CBS_OWNERDRAWVARIABLE*** があり ***CBS_HASSTRINGS*** のないコンボボックスの場合，コンボボックスのオーナーに対して ***WM_DELETEITEM*** メッセージが項目ごとに送られます。

CB_SELECTSTRING

コンボボックスのリストボックスから，該当するプレフィクス文字列を持つ最初の項目を選択します。エディットコントロールまたはスタティックテキストコントロールは，選択項目に応じて更新されます。

引数 *wParam*

検索開始位置のインデックス。***wParam*** の次の項目から検索が開始されます。検索がリスト末尾におよぶと，続けて項目0から検索が行なわれます。***wParam*** に戻れば終了します。***wParam*** が-1の場合はリスト全体が項目0から検索されます。

lParam

プレフィクス文字列（ヌルで終わる文字列）。

戻り値 成功すれば最初の該当項目のインデックス，失敗すれば ***CB_ERR*** が返され，現在の選択項目は変更されません。

コメント *CBS_OWNERDRAWFIXED* か *CBS_OWNERDRAWVARIABLE* があり *CBS_HASSTRINGS* のないコンボボックスの場合，*lParam* は32ビット値となりリスト内の対応する各32ビット値と比較されます。

CB_SETCURSEL

コンボボックスのリストボックスから項目を選択します。さらに選択項目に合わせてエディットコントロールまたはスタティックテキストコントロールを更新します。

引数 *wParam*

項目のインデックス。-1であれば項目の選択は行なわれません。

lParam

ありません。

戻り値 *wParam* が-1または無効なインデックスの場合は ***CB_ERR***，その他の場合は選択項目のインデックス。

CB_SETEDITSEL

コンボボックスのエディットコントロール内の選択テキストをセットします。

引数 *wParam*

ありません。

LOWORD *(lParam)*

開始文字のインデックス。

HIWORD *(lParam)*

終了文字のインデックス。

戻り値 成功すれば0以外，失敗すれば0。エディットコントロールのないコンボボックスの場合には ***CB_ERR***。

CB_SETITEMDATA

コンボボックスのリストボックスの項目に対応する32ビット値をセットします。

引数 *wParam*

項目のインデックス。

lParam

項目に設定する新しい32ビット値。

戻り値 エラーの場合には ***CB_ERR***。

CB_SHOWDROPDOWN

コンボボックスのドロップダウンリストボックスを，表示または非表示にします。

引数 *wParam*

0の場合はドロップダウンリストボックスが非表示，その他の場合は表示されます。

lParam

ありません。

戻り値 ありません。

コメント このメッセージが有効なのは，***CBS_DROPDOWN*** または ***CBS_DROPDOWNLIST*** スタイルのコンボボックスだけです。

DM_GETDEFID

ダイアログのデフォルトプッシュボタンコントロールIDを返します。

引数 *wParam*

ありません。

lParam
ありません。

戻り値 ダイアログにデフォルトプッシュボタンIDがなければ，戻り値の上位ワードは0。デフォルトがあれば，戻り値の上位ワードは ***DC_HASDEFID***，下位ワードはデフォルトプッシュボタンID。

DM_SETDEFID

ダイアログのプッシュボタンコントロールIDをセットします。

引数 *wParam*
新しいデフォルトプッシュボタンID。

lParam
ありません。

戻り値 ありません。

EM_CANUNDO

エディットコントロールが，***EM_UNDO*** メッセージに正しく応答できるかどうかを調べます。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ***EM_UNDO*** メッセージに応答できるエディットコントロールの場合は0以外，できない場合は0。

EM_EMPTYUNDOBUFFER

エディットコントロールのアンドゥバッファをクリアして，前回の編集をやり直しできないようにします。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント ***WM_SETTEXT*** または ***EM_SETHANDLE*** がエディットコントロールに送られると，エディットコントロールのアンドゥバッファは自動的にクリアされます。

EM_FMTLINES

特殊な行末文字シーケンスを，ワードラップ改行テキストに追加するかどうかを，エディットコントロールに指示します。

引数 *wParam*

0以外の場合，テキスト行末のワードラップは，キャリッジリターン，キャリッジリターン，改行文字のシーケンスで行なわれます。0の場合には，このキャリッジリターン，キャリッジリターン，改行文字のシーケンスはテキストからすべて除去されます。

lParam

ありません。

戻り値 テキストが変更されると0以外，変更がなければ0。

コメント キャリッジリターン1回と改行文字による通常の改行に対しては，このメッセージは効力を持ちません。0以外の戻り値の場合はテキストサイズが変わります。このメッセージを使えるのは複数行のエディットコントロールに限られます。

EM_GETHANDLE

エディットコントロールのバッファのハンドルを返します。バッファにはエディットコントロールのテキストが含まれます。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 エディットコントロールのバッファのハンドル。

コメント このメッセージの送り先は，***DS_LOCALEDIT*** で作成されたエディットコントロールに限られます。

EM_GETLINE

エディットコントロールの1行を返します。

引数 *wParam*
行番号。エディットコントロールの行は0から始まります。

lParam
行の格納先となるバッファへのポインタ。バッファの最初のワードは，バッファに転送されるバイト数です。

戻り値 実際に転送されたバイト数。バッファにはヌル文字が不要です。

コメント このメッセージは複数行のエディットコントロールにのみ有効です。

EM_GETLINECOUNT

エディットコントロール内のテキスト行数を返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 テキストの行数。

コメント 複数行のエディットコントロールにのみ有効です。

EM_GETMODIFY

エディットコントロールの変更フラグを返します。変更フラグがセットされるのは，エディットコントロールへの新しいテキストの入力，既存テキストの変更，***EM_SETMODIFY*** メッセージ受け取りの場合です。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 エディットコントロールの変更フラグ。0 以外はテキストの変更，0 は無変更を示します。

EM_GETRECT

エディットコントロールの書式化矩形を取得します。

引数 *wParam*

ありません。

lParam

このメッセージを格納する ***RECT*** データ構造体へのポインタ。

戻り値 ありません。

EM_GETSEL

エディットコントロール内の選択テキストの，最初と最後のインデックスを返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 戻り値の下位ワードは開始インデックス，上位ワードは終了インデックス。

EM_LIMITTEXT

エディットコントロールへの入力文字数を制限します。

引数 *wParam*

最大許容文字数。0 であれば文字数の制限はありません。

lParam

ありません。

戻り値 成功すれば 0 以外，失敗すれば 0。

EM_LINEFROMCHAR

指定文字のインデックスを含むエディットコントロール内の行の番号を返します。

引数 *wParam*

エディットコントロール内の指定文字のインデックス，または -1。

lParam

ありません。

戻り値 *wParam* が -1 の場合は，選択テキスト内で最初の文字を含む行の行番号。-1 以外の場合は *wParam* 文字インデックスを含む行番号。

EM_LINEINDEX

エディットコントロール内の行の先頭文字インデックスを返します。

引数 *wParam*

行番号。*wParam* が -1 の場合はキャレット（^）が現在位置する行が使用されます。

lParam

ありません。

戻り値 該当行の先頭文字インデックス。

コメント 複数行のエディットコントロールのみに有効です。

EM_LINELENGTH

指定文字インデックスを含むエディットコントロール内の行のバイト長を返します。

引数 *wParam*
エディットコントロール内の文字のインデックス，または - 1。

lParam
ありません。

戻り値 *wParam* が - 1 であればキャレットがある行の長さ，- 1 でない場合は文字インデックス *wParam* を含む行の長さ。

コメント 選択が行なわれているテキスト部分は，その行数にかかわらず対象外として，このメッセージから無視されます。したがって，行の長さには含まれません。

EM_LINESCROLL

エディットコントロールをスクロールします。

引数 *wParam*
ありません。

LOWORD *(lParam)*
垂直スクロールされる行数です。

HIWORD *(lParam)*
水平スクロールされる文字数です。

戻り値 ありません。

コメント 複数行のエディットコントロールのみに有効です。

EM_REPLACESEL

エディットコントロール内の指定テキストを置き換えます。

引数 *wParam*

ありません。

lParam

現在選択中のテキストと置き換える，ヌルで終わるテキストへのポインタです。

戻り値 ありません。

EM_SETHANDLE

エディットコントロールのテキストバッファをセットします。

引数 *wParam*

エディットコントロールのテキストバッファへのローカルハンドル。

lParam

ありません。

戻り値 ありません。

コメント 以前のテキストバッファを ***EM_GETHANDLE*** で取得して ***LocalFree*** 関数で破棄します。その後に ***EM_SETHANDLE*** メッセージで新しいテキストバッファを用意します。複数行のエディットコントロールのみに有効なメッセージです。

EM_SETMODIFY

エディットコントロールの変更フラグをセットします。

引数 *wParam*

新しい変更フラグ値。

lParam

ありません。

戻り値 ありません。

EM_SETPASSWORDCHAR

ES_PASSWORD で作成されたエディットコントロールに入力された文字のかわりに表示される文字を設定します。

引数 *wParam*

新しい表示文字，または0。0の場合は実際に入力された文字が表示されます。

lParam

ありません。

戻り値 ありません。

EM_SETRECT

エディットコントロールのために書式化された矩形領域を設定します。それに応じてテキストも再表示します。

引数 *wParam*

ありません。

lParam

新しく書式化された矩形領域を指定する ***RECT*** 構造体へのポインタ。

戻り値 ありません。

コメント 複数行のエディットコントロールのみに有効です。

EM_SETRECTNP

エディットコントロールのために書式化された矩形領域を設定します。テキストの再表示は行ないません。

引数 *wParam*

ありません。

lParam

新しく書式化された矩形領域を指定する ***RECT*** 構造体へのポインタ。

戻り値 ありません。

コメント テキストを後で書き直すときは，このメッセージを ***EM_SETRECT*** のかわりに使います。ただし複数行のエディットコントロールのみに有効です。

EM_SETSEL

エディットコントロールに選択テキストをセットします。

引数 *wParam*

ありません。

LOWORD *(lParam)*

開始文字のインデックス。

HIWORD *(lParam)*

終了文字のインデックス。

戻り値 ありません。

EM_SETTABSTOPS

エディットコントロールのタブストップをセットします。

引数 *wParam*

1, タブストップ数, 0のいずれか。

lParam

wParam も *lParam* もともに0であれば, タブストップ位置は32ダイアログ単位ごとになります。*wParam* が1であれば, タブストップ位置は *lParam* ダイアログ単位ごとに等間隔に設定されます。*wParam* が0でも1でもない場合, *lParam* は *wParam* 個以上の要素を含む整数型配列へのポインタになります。配列上の各要素は, ダイアログ単位によるタブストップ位置を昇順で示します。

戻り値 すべてのタブストップがセットされれば0以外, そうでなければ0。

コメント ダイアログ単位は現在のダイアログ基本幅単位の4分の1です。基本幅単位は ***GetDailogBaseUnits*** 関数によって得られます。このメッセージは複数行のエディットコントロールのみに有効です。

EM_SETWORDBREAK

エディットコントロールのワードブレーク関数を変更します。

引数 *wParam*

ありません。

lParam

ワードブレーク関数の手続きインスタンスアドレス。***MakeProcInstance*** によって作成されます。ワードブレーク関数の宣言は下記のように行ないます。

function LPSTR WordBreakFunction (LPSTR EditText, int Current Word, int EditTextCount) ;

WordBreakFunction はリテラル名称ではないので，別名でもかまいません。***EditText*** はエディットコントロールのテキストのポインタです。引数 *CurrentWord* はテキスト内の最初のワードへのインデックスです。*EditTextCount* はテキストの総バイト数です。ワードブレーク関数は，テキストの次のワードの先頭文字へのポインタを返すものとします。テキストの最後のワードの場合は，テキストの最終文字の直後の文字へのポインタを返すものとします。

戻り値 ありません。

コメント ワードブレーク関数のデフォルト定義では，空白文字列に続く最初の空白以外の文字が次のワードの先頭文字です。

EM_UNDO

エディットコントロールのテキストの最終変更をします。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 成功すれば0以外。失敗すれば0が返されて，エディットコントロールのテキストは変更されません。

コメント エディットコントロールのテキストに加えた変更は，すべてアンドゥバッファに格納されます。このメッセージが無効になるとすれば，その原因はメモリ不足です。アンドゥ操作を行なうだけの充分なアンドゥバッファがない場合に限られます。

LB_ADDSTRING

リストボックスに文字列を加えます。

引数 *wParam*

ありません。

lParam

追加文字列（ヌルで終わる文字列）へのポインタ。

戻り値 成功すれば文字列が追加された位置のインデックス，メモリ不足で文字列が格納されなかった場合は ***LB_ERRSPACE***，エラーの場合は ***LB_ERR***。

コメント ソートされないリストボックスの場合，文字列の追加位置はリスト末尾になります。***LBS_OWNERDRAWFIXED*** か ***LBS_OWNERDRAWVARIABLE*** があり ***LBS_HASSTRINGS*** のないリストボックスの場合，*lParam* は32ビット値となり文字列のかわりに格納されます。さらにリストボックスのオーナーに ***WM_COMPAREITEM*** メッセージが送られ，各追加項目と他項目との比較が行なわれます。

LB_DELETESTRING

リストボックスから文字列を削除します。

引数 *wParam*

削除する項目のインデックス。

lParam

ありません。

戻り値 *wParam* が有効なインデックスであればリストの残存項目数，無効なインデックスであれば ***LB_ERR***。

コメント ***LBS_OWNERDRAWFIXED*** か ***LBS_OWNERDRAWVARIABLE*** があり ***LBS_HASSTRINGS*** のないリストボックスの場合，対応する 32 ビット値が削除されます。リストボックスのオーナーには ***WM_DELETEITEM*** メッセージが送られます。

LB_DIR

カレントディレクトリ内のファイル名で，ファイル仕様と DOS ファイル属性に一致するものを，リストボックスに加えます。

引数 *wParam*

DOS ファイル属性。

lParam

ファイル仕様文字列（ヌルで終わる文字列）へのポインタ。

戻り値 成功すればファイル名追加後のリストの末尾項目のインデックス，メモリ不足で項目の格納ができない場合は ***LB_ERRSPACE***，エラーの場合は ***LB_ERR***。

LB_FINDSTRING

プレフィクス文字列に最初に一致するリストボックス項目を検索します。

引数 *wParam*

検索開始位置のインデックス。*wParam* の次の項目から検索が開始されます。検索がリストの末尾までくると，続けて項目 0 から検索が行なわれます。*wParam* に戻れば終了します。*wParam* が -1 の場合はリスト全体が項目 0 から検索されます。

lParam

プレフィクス文字列（ヌルで終わる文字列）。

戻り値 成功すれば最初に一致した項目のインデックス，失敗すれば ***LB_ERR***。

コメント ***LBS_OWNERDRAWFIXED*** か ***LBS_OWNERDRAWVARIABLE*** スタイルがあり ***LBS_HASSTRINGS*** スタイルのないリストボックスの場合，*lParam* は 32 ビット値となり，リスト内の対応する 32 ビット値と比較されます。

L

Ms

LB_GETCOUNT

リストボックスの項目数を返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 リストボックスの項目数。

LB_GETCURSEL

リストボックス内で現在選択されている項目のインデックスを返します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 選択されている項目がなければ ***LB_ERR***，あれば現在選択されている項目のインデックス。

LB_GETHORIZONTALEXTENT

リストボックスの水平スクロール可能な幅をピクセル数で返します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 リストボックスの水平スクロール可能な幅のピクセル数。

コメント ***WS_HSCROLL*** スタイルで作成されたリストボックスのみに有効です。

LB_GETITEMDATA

リストボックス項目に対応する 32 ビット値を返します。

引数 *wParam*
項目のインデックス。

lParam

ありません。

戻り値　成功すれば対応する32ビット値，失敗すれば ***LB_ERR***。

LB_GETITEMRECT

リストボックス項目を表示する際の矩形枠を取得します。

引数　*wParam*

項目のインデックス。

lParam

矩形値を入れる ***RECT*** 構造体へのポインタ。

戻り値　エラーの場合は ***LB_ERR***。

LB_GETSEL

リストボックス項目の選択の有無を返します。

引数　*wParam*

項目のインデックス。

lParam

ありません。

戻り値　エラーの場合は ***LB_ERR***。項目の選択があれば正の値，選択がなければ0。

LB_GETSELCOUNT

リストボックス内で現在選択されている項目の数を返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 複数項目を同時に選択できるリストボックスの場合は選択中の項目数，その他の場合は ***LB_ERR***。

LB_GETSELITEMS

リストボックス内で現在選択されている項目のインデックスを取得します。

引数 *wParam*

取得する項目インデックスの最大値。

lParam

wParam の項目インデックスを保持する容量のある整数型配列へのポインタ。

戻り値 複数項目を同時に選択できるリストボックスの場合は，*wParam* で指定した項目数のインデックスが *lParam* の配列に入ります。その配列内の指定項目総数が返されます。その他の場合は ***LB_ERR*** です。

LB_GETTEXT

リストボックス項目を受け取るバッファにコピーします。

引数 *wParam*

項目のインデックス。

lParam

バッファへのポインタ。バッファは項目の末尾にヌル文字をつけて格納できる容量とします。

戻り値 ありません。

コメント ***LBS_OWNERDRAWFIXED*** か ***LBS_OWNERDRAWVARIABLE*** はあるが ***LBS_HASSTRINGS*** のないリストボックスの場合，リスト項目に対応する 32 ビット値がバッファにコピーされます。

LB_GETTEXTLEN

リストボックス項目の長さをバイト数で返します。

引数 *wParam*

項目のインデックス。

lParam

ありません。

戻り値 *wParam* が有効なインデックスであればこのインデックス位置にある項目の長さ，無効なインデックスであれば ***LB_ERR***。

LB_GETTOPINDEX

リストボックスの先頭に表示された項目のインデックスを返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 リストボックスの先頭に表示された項目のインデックス。

コメント 初期状態では，項目0がリストボックスの先頭表示項目です。リストボックスがスクロールされれば，別の項目が先頭にきます。

LB_INSERTSTRING

文字列をソートせずにリストボックスに挿入します。

引数 *wParam*

-1の場合はリスト末尾に文字列が挿入されます。-1以外は文字列挿入位置のインデックスとなります。

lParam

挿入文字列（ヌルで終わる文字列）へのポインタ。

戻り値 成功すれば文字列が挿入された位置のインデックス，メモリ不足で文字列を格納できなかった場合は ***LB_ERRSPACE***，エラーの場合は ***LB_ERR***。

LB_RESETCONTENT

リストボックスから項目すべてを除去します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント ***LBS_OWNERDRAWFIXED*** か ***LBS_OWNERDRAWVARIABLE*** はあるが ***LBS_HASSTRINGS*** のないコンボボックスの場合，***WM_DELETEITEM*** メッセージがリストボックスのオーナーに送られます。

L
Msg

LB_SELECTSTRING

プレフィクス文字列に最初に一致するリストボックス項目を検索します。

引数 *wParam*

検索開始位置のインデックス。***wParam*** の次の項目から検索が開始されます。検索がリストの末尾にくると，続けて項目 0 から検索が行なわれます。***wParam*** に戻れば終了します。***wParam*** が −1 の場合は，リスト全体が項目 0 から検索されます。

lParam

プレフィクス文字列（ヌルで終わる文字列)。

戻り値 成功すれば最初に一致した項目のインデックス。失敗すれば ***LB_ERR*** が返され，現在の選択項目に変更はありません。

コメント ***LBS_OWNERDRAWFIXED*** か ***LBS_OWNERDRAWVARIABLE*** はあるが ***LBS_HASSTRINGS*** のないリストボックスの場合，***lParam*** は 32 ビット値となり，リスト内の対応 32 ビット値との比較が行なわれます。

LB_SELITEMRANGE

リストボックス内の連続する項目を，選択または選択解除します。

引数 *wParam*

0の場合は項目の選択解除が行なわれます。0以外の場合は選択が行なわれます。

LOWORD *(lParam)*

最初の項目のインデックス。

HIWORD *(lParam)*

最後の項目のインデックス。

戻り値 エラーの場合は ***LB_ERR***。

コメント 複数選択型のリストボックスのみに有効です。

LB_SETCOLUMNWIDTH

リストボックスの桁の幅をセットします。

引数 *wParam*

各桁の幅（ピクセル単位)。

lParam

ありません。

戻り値 ありません。

コメント ***LBS_MULTICOLUMN*** スタイルのリストボックスのみに有効です。

LB_SETCURSEL

リストボックス項目を選択します。

引数 *wParam*

項目のインデックス。-1であれば何も選択されません。

lParam

ありません。

戻り値 *wParam* が-1または無効なインデックスであれば ***LB_ERR***，その他の場合は選択された項目のインデックス。

LB_SETHORIZONTALEXTENT

リストボックスの水平スクロール可能な幅（ピクセル単位）をセットします。

引数 *wParam*

リストボックスの水平スクロール可能な幅（ピクセル単位）。

lParam

ありません。

コメント ***WS_HSCROLL*** スタイルで作成されたリストボックスのみに有効です。リストボックスの幅が指定した幅より狭ければ水平スクロールバーが有効になり，等しいか広ければ無効になります。

LB_SETITEMDATA

リストボックス項目に対応した32ビット値をセットします。

引数 *wParam*

項目のインデックス。

lParam

項目に対応する新しい32ビット値。

戻り値 エラーの場合は ***LB_ERR***。

LB_SETSEL

リストボックス項目を選択または選択解除します。

引数 *wParam*

0であれば項目は選択解除され，0以外であれば項目は選択されます。

***LOWORD** (lParam)*

-1であればリストボックス内の全項目が対象となり，-1以外のときは *lParamLo* で対象項目を決定します。

***HIWORD** (lParam)*

lParam が-1でないときに項目のインデックスになります。

戻り値 エラーの場合は ***LB_ERR***。

コメント 複数選択型のリストボックスのみに有効です。

LB_SETTABSTOPS

リストボックスにタブストップをセットします。

引数 *wParam*

1, タブストップ数，0のいずれか。

lParam

wParam も ***lParam*** もともに0であれば，タブストップ位置は2ダイアログ単位ごとになります。***wParam*** が1であれば，タブストップ位置は ***lParam*** ダイアログ単位ごとに等間隔に設定されます。***wParam*** が0でも1でもない場合，***lParam*** は ***wParam*** 個以上の要素を含む整数型配列へのポインタになります。 配列上の各要素は，ダイアログ単位によるタブストップ位置を昇順で示します。

戻り値 すべてのタブストップがセットされれば0以外，そうでなければ0。

コメント ダイアログ単位は現在のダイアログ基本幅単位の4分の1です。基本幅単位は ***GetDialogBaseUnits*** 関数によって得られます。このメッセージは複数選択型のリストボックスのみに有効です。

LB_SETTOPINDEX

リストボックスの先頭に表示される項目のインデックスをセットします。

引数 *wParam*

項目のインデックス。

lParam

ありません。

戻り値 エラーの場合は ***LB_ERR***。

WM_ACTIVATE

ウィンドウにアクティブ化または非アクティブ化を通知します。

引数 *wParam*

ウィンドウが非アクティブ化されると0になります。ウィンドウのアクティブ化がマウスクリック以外によって行なわれると1になります。ウィンドウのアクティブ化がマウスクリックによって行なわれると2になります。

HIWORD *(lParam)*

ウィンドウが最小化されていれば0以外，最小化されていなければ0。

LOWORD *(lParam)*

wParam が0の場合はアクティブ化されるウィンドウへのハンドル，0以外の場合は非アクティブ化されるウィンドウへのハンドル。

戻り値 ありません。

コメント ウィンドウが最少化されておらず，アクティブの場合，DefWindowProc はウィンドウ入力フォーカスを渡します。

WM_ACTIVATEAPP

アクティブウィンドウがあるアプリケーションから別のアプリケーションに切り換わるときに，その2つのアプリケーションに対してその旨を通知します。

引数 *wParam*

0にすると別のアプリケーションのウィンドウがアクティブ化されます。0以外にすると現アプリケーションのウィンドウがアクティブになります。

lParam

他アプリケーションへのタスクハンドル。

戻り値 ありません。

WM_ASKCBFORMATNAME

クリップボードのオーナーに対しクリップボードのデータ書式名を要求します。

引数 *wParam*

lParam バッファにコピー可能な最大名称長。

lParam

書式名をコピーするバッファへのポインタ。

戻り値 ありません。

コメント クリップボード書式が ***CF_OWNERDISPLAY*** のときに，クリップボードのオーナーに送られるメッセージは以下のものがあります。

WM_ASKCBFORMATNAME, WM_HSCROLLCLIPBOARD, WM_PAINTCLIPBOARD, WM_SIZECLIPBOARD, WM_VSCROLLCLIPBOARD

クリップボードデータと書式は ***SetClipboardData*** 関数によってセットします。

WM_CANCELMODE

メッセージボックスを表示して，システムの現在のモードを取り消すことを通知します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント このメッセージはアプリケーションに対し，マウスまたはキーボード入力の送り先がメッセージボックスになることを警告します。メッセージボックスを除去すると，キーボードとマウスボタンの状態，またはマウス位置が把握できなくなります。

WM_CHANGECBCHAIN

クリップボードチェインからウィンドウを除去することを，クリップボードチェインの最初のウィンドウに通知します。

引数 *wParam*

クリップボードチェインから除去されるウィンドウへのハンドル。

LOWORD *(lParam)*

除去されるウィンドウの次の位置にあるウィンドウへのハンドル。

HIWORD *(lParam)*

ありません。

戻り値 ありません。

コメント ***SendMessage*** 関数で，このメッセージをクリップボードチェインの次のウィンドウに送る必要があります。このウィンドウのハンドルの初期値は，***SetClipboardViewer*** 関数がウィンドウをチェインへ追加したときの戻り値です。このウィンドウの保管ハンドルと *wParam* が等しい場合は，次のウィンドウの新ハンドルは，*LOWORD (lParam)* になります。***WM_DESTROY*** メッセージを受け取ったウィンドウは，クリップボードチェインから離れるものとします。***WM_DRAWCLIPBOARD*** も参照してください。

WM_CHAR

フォーカスのあるウィンドウに，非システムキーが押されたことを通知します。

引数 *wParam*

キー値。

LOWORD *(lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

***HIWORD** (lParam)*
ビット0から7まではOEM依存のキー走査コードです。拡張キーの場合はビット8が1です。***GRPH*** の押下中に別のキーの押下が行なわれればビット13は1です。メッセージ転送前にキーが押下状態であればビット14は1です。ビット15はキーを離す場合は1に，押した場合は0になります。

戻り値 ありません。

コメント 非システムキーとは，***GRPH*** が押下状態でないときに押されたキーのことです。アプリケーションに普通必要なのは，***LOWORD** (lParam)* と ***HIWORD** (lParam)* のビット0から7までです。フォーカスのあるウィンドウがなければ，***WM_SYSKEYDOWN, WM_SYSCHAR, WM_SYSKEYUP*** の各メッセージが ***WM_KEYDOWN, WM_CHAR, WM_KEYUP*** のかわりに送られます。***WM_DEADCHAR, WM_SYSCHAR, WM_SYSDEADCHAR*** も参照してください。

WM_CHARTOITEM

リストボックスのオーナー対し，***WM_CHAR*** メッセージに対する応答を要求します。

引数 *wParam*
押したキー。

***LOWORD** (lParam)*
リストボックスへのハンドル。

***HIWORD** (lParam)*
現在のキャレット位置。

戻り値 －2が返されれば，アプリケーションがすべての処理を行なっています。－1が返されれば，リストボックスがキーに応じてデフォルト動作を実行します。0以上が返されれば，リストボックスは戻り値で指定する項目へのキーに応じてデフォルト動作を実行します。

コメント このメッセージは ***LBS_WANTKEYBOARDINPUT*** スタイルのリストボックスに限って有効です。***WM_VKEYTOITEM*** も参照してください。

WM_CHILDACTIVATE

SetWindowPos 関数で子ウィンドウを移動したことを親ウィンドウに通知します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

WM_CLEAR

ウィンドウの現在の選択内容を削除します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

WM_CLOSE

ウィンドウにクローズを通知します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

コメント ***DefWindowPrc*** 関数のデフォルト動作では，***DestroyWindow*** 関数を呼び出してウィンドウを破棄します。

WM_COMMAND

メニュー項目の選択，アクセラレータキーの変換，子コントロールからのメッセージの転送をウィンドウに通知します。

引数 *wParam*
メニュー項目，アクセラレータ ID，コントロール ID のいずれか。

LOWORD *(lParam)*
0 の場合はメッセージの転送元がメニューであり，***HIWORD*** *(lParam)* は使用されません。0 以外の場合，***LOWORD*** *(lParam)* の値は ***HIWORD*** *(lParam)* によって異なります。

HIWORD *(lParam)*
1 の場合は ***LOWORD*** *(lParam)* がアクセラレータ ID になります。1 でなければ ***LOWORD*** *(lParam)* は子コントロールへのハンドルとなり，***HIWORD*** *(lParam)* は子コントロールによって送られるメッセージの通知コードになります。詳細については第 7 章 **『Windows 定数』** の「**BN_ボタン通知コード**」,「**EN_エディットコントロール通知コード**」,「**LBN_リストボックス通知コード**」,「**CBN_コンボボックス通知コード**」を参照してください。

戻り値 ありません。

コメント システムメニュー項目に割り当てられるアクセラレータキーは，***WM_COMMAND*** メッセージではなくて ***WM_SYSCOMMAND*** メッセージに変換されます。***WM_COMMAND*** がアクセラレータに送られるのは，ウィンドウが最小化されていないときと，最小化されていてもアクセラレータに一致する項目がウィンドウメニューまたはシステムメニューにないときです。

WM_COMPACTING

メモリ圧縮時間がシステムタイムの12.5パーセントを超えることを，最上位ウィンドウに通知します。

引数 *wParam*

CPU時間に占めるメモリ圧縮時間のパーセンテージを示し，65,535は100パーセントを表わします。たとえば，*wParam* が32,768であれば，メモリ圧縮時間はCPU時間の50パーセントです。

lParam

ありません。

戻り値 ありません。

コメント このメッセージが送られると，アプリケーションはメモリをできるだけ解放します。リソースの現使用状況や使用中のアプリケーション総数を配慮してください。***GetNumTasks*** 関数でアプリケーション数が取得できます。

WM_COMPAREITEM

2つの項目を比較してソート順を示す値を返すことを，オーナー描画のコンボボックスまたはリストボックスのオーナーに要求します。

引数 *wParam*

ありません。

lParam

COMPAREITEMSTRUCT 構造体へのポインタ。この構造体には両項目の識別子とデータが含まれています。

戻り値 項目1を前に置く場合は -1, 同値の場合は 0, 項目2を前に置く場合は 1。

コメント このメッセージの対象は次に限られます。コンボボックスでは，***CBS_SORT*** スタイルと ***CBS_OWNERDRAWFIXED*** か ***CBS_OWNERDRAWVARIABLE*** を持つものです。リストボックスでは，***LBS_SORT*** スタイルと ***LBS_OWNERDRAWFIXED*** か ***LBS_OWNERDRAWVARIABLE*** スタイルを持つものです。

WM_COPY

現選択内容を ***CF_TEXT*** 書式でクリップボードにコピーします。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

WM_CREATE

ウィンドウの作成が終了したので初期化が必要であることをウィンドウに通知します。

引数 *wParam*
ありません。

lParam
CreateWindow 関数に渡される情報を含む ***CREATESTRUCT*** 構造体へのポインタです。

戻り値 ありません。

コメント このメッセージが送られるのは，***CreateWindow*** 関数呼び出しのときで，ウィンドウのオープン前です。

WM_CTLCOLOR

子コントロールまたはメッセージボックスの親ウィンドウに対し，子ウィンドウのテキスト色とバックグラウンド色の変更を可能にします。

引数 *wParam*

子ウィンドウのディスプレイコンテキストへのハンドル。

***LOWORD** (lParam)*

子ウィンドウへのハンドル。

***HIWORD** (lParam)*

CTLCOLOR_定数のいずれか。子ウィンドウの種別を指定します。第7章『**Windows 定数**』の「**CTLCOLOR_コントロールカラーフラグ**」を参照してください。

戻り値 ありません。

コメント ***DefWindowProc*** 関数のデフォルト動作では，デフォルトのシステムカラーが使用されます。

WM_CUT

現在の選択内容を ***CF_TEXT*** 書式でクリップボードにコピー後，削除します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

WM_DDE_ACK

別の DDE メッセージを受け取ったことをアプリケーションに通知します。

引数 *wParam*

メッセージ転送元のウィンドウのハンドル。

lParam

転送メッセージが ***WM_DDE_INITIATE*** の場合，***LOWORD*** *(lParam)* は応答アプリケーションを指名するアトムを，***HIWORD*** *(lParam)* は応答サーバウィンドウに対応するトピックを含むアトムを保持します。転送メッセージが ***WM_DDE_EXECUTE*** の場合，***LOWORD*** *(lParam)* は応答のステータスを示すレコードを，***HIWORD*** *(lParam)* はコマンド文字列を含むデータ項目へのハンドルを保持します。その他のメッセージに関しては，***LOWORD*** *(lParam)* がステータスレコードを，***HIWORD*** *(lParam)* が応答送り先のデータ項目を指定するアトムを保持します。

コメント このメッセージは必ず ***SendMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第1引数にしてください。

WM_DDE_ADVISE

データ項目の変更のたびに更新を行なうことを要求するメッセージで，クライアントアプリケーションからサーバ（受信側）アプリケーションに送られます。

引数 *wParam*

送信側ウィンドウのハンドル。

LOWORD *(lParam)*

データ転送方法を示す ***DDEADVISE*** 構造体。

HIWORD *(lParam)*

要求データ項目を指定するアトム。

コメント このメッセージは必ず ***PostMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第1引数にしてください。

WM_DDE_DATA

サーバアプリケーションから送られるメッセージです。データ項目値の転送を行ないます。または項目の有無をクライアントに知らせます。

引数 *wParam*

送信側ウィンドウのハンドル。

***LOWORD** (lParam)*

DDEDATA 構造体をデータ格納場所とするグローバルメモリブロックへのハンドル。単なる変更通知メッセージの場合は 0。

***HIWORD** (lParam)*

転送データ項目を指定するアトム。

コメント このメッセージは必ず ***PostMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第1引数にしてください。

WM_DDE_EXECUTE

クライアントアプリケーションから一連のコマンドをサーバアプリケーションに送って処理します。

引数 *wParam*

送信側ウィンドウのハンドル。

***LOWORD** (lParam)*

予約されています。

***HIWORD** (lParam)*

コマンドを含むグローバルメモリオブジェクトへのハンドル。

コメント このメッセージは必ず ***PostMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第1引数にしてください。

WM_DDE_INITIATE

会話を始動するメッセージで，クライアントまたはサーバから送られます。応答側アプリケーションは ***WM_DDE_ACK*** メッセージで応答します。

引数 *wParam*

送信側ウィンドウのハンドル。

LOWORD *(lParam)*

会話の相手とするアプリケーション名を指定するアトム。どのアプリケーションでもかまわない場合は 0。

HIWORD *(lParam)*

会話の内容とするトピックを指定するアトム。どんなトピックでもかまわない場合は 0。

コメント このメッセージは必ず ***SendMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第 1 引数にしてください。

WM_DDE_POKE

非要求データの受け取りをサーバアプリケーションに要求するメッセージで，クライアントアプリケーションから送られます。サーバは ***WM_DDE_ACK*** メッセージで応答します。

引数 *wParam*

送信側ウィンドウのハンドル。

LOWORD *(lParam)*

DDEPOKE 構造体へのハンドル。

HIWORD *(lParam)*

データ項目を識別するアトム。

コメント このメッセージは必ず ***PostMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第 1 引数にしてください。

WM_DDE_REQUEST

特定データ項目の値を要求するメッセージで，クライアントアプリケーションから送られます。

引数 *wParam*

送信側ウィンドウのハンドル。

LOWORD *(lParam)*

クリップボード書式番号（第7章『**Windows 定数**』の「**CF_クリップボード書式**」を参照）。

HIWORD *(lParam)*

要求データ項目を識別するアトム。

コメント このメッセージは必ず ***PostMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第1引数としてください。

WM_DDE_TERMINATE

会話終了のメッセージで，アプリケーションから送られます。

引数 *wParam*

送信側ウィンドウのハンドル。

lParam

予約されています。

コメント このメッセージは必ず ***PostMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第1引数にしてください。

WM_DDE_UNADVISE

特定項目またはそのクリップボード書式の更新が不要になったことを，サーバに通知するメッセージです。クライアントアプリケーションからサーバアプリケーションに送られます。

引数 *wParam*

送信側ウィンドウのハンドル。

LOWORD *(lParam)*

クリップボード書式番号（第7章『**Windows 定数**』の「**CF_クリップボード書式**」を参照)。

HIWORD *(lParam)*

データ項目を識別するアトム。

コメント このメッセージは必ず ***PostMessage*** 関数で送ります。メッセージ受信側ウィンドウのハンドルを第1引数にしてください。

WM_DEADCHAR

デッド文字をウィンドウに通知します。

引数 *wParam*

キー値。

***LOWORD** (lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

***HIWORD** (lParam)*

ビット 0 から 7 までは OEM 依存のキー走査コードです。拡張キーの場合はビット 8 が 1 です。**GRPH** キーの押下中にキーの押下が行なわれればビット 13 は 1 です。メッセージ転送前にキーが押下状態であればビット 14 は 1 で，繰り返し数は正の数になります。ビット 15 はキーを離す場合は 1 に，押す場合には 0 になります。

戻り値 ありません。

コメント デッドキーには，ウムラウト文字とアクセント文字が含まれます。押しても単独では文字にならないキーの場合，このメッセージでフィードバックを行なうことができます。アプリケーションに普通必要なのは，***LOWORD** (lParam)* と ***HIWORD** (lParam)* のビット 0 から 7 までです。***WM_CHAR**, **WM_SYSCHAR**, **WM_SYSDEADCHAR*** も参照してください。

WM_DELETEITEM

リストボックス項目が削除されたことを，リストボックスまたはコンボボックスに通知します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント このメッセージの対象は次に限られます。コンボボックスでは，***CBS_OWNERDRAWFIXED*** か ***CBS_OWNERDRAWVARIABLE*** のスタイルを持つものです。リストボックスでは，***LBS_OWNERDRAWFIXED*** か ***LBS_OWNERDRAWVARIABLE*** のスタイルを持つものです。このメッセージが送られるのは，コンボボックスまたはリストボックスが破棄された場合と，項目の削除が ***LB_DELETESTRING***, ***LB_RESETCONTENT***, ***CB_DELETESTRING***, ***CB_RESETCONTENT*** の各メッセージによって行なわれた場合です。

WM_DESTROY

ウィンドウに対し破棄を通知します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

コメント クリップボードチェインにあるウィンドウは，***WM_DESTROY*** メッセージ処理を終了する前に，必ず ***ChangeClipboardChain*** 関数で自分をチェインから外す責任があります。***WM_DESTROY*** メッセージは，ウィンドウが画面から除去されると，***DestroyWindow*** 関数から送られるものです。子ウィンドウが消去される前に，親ウィンドウにメッセージが送られます。

WM_DESTROYCLIPBOARD

EmptyClipboard 関数呼び出しでクリップボードが空になったことを，クリップボードオーナーに通知します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

WM_DEVMODECHANGE

デバイスモードの設定を変更したことを各最上位ウィンドウに通知します。

引数 *wParam*
ありません。

lParam
デバイス名へのポインタ。

戻り値 ありません。

コメント デバイス名は Windows 初期化ファイルの WIN.INI にある文字列です。

WM_DRAWCLIPBOARD

クリップボードの内容が変わったことをクリップボードチェイン上の最初のウィンドウに通知します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント ***SendMessage*** 関数で，このメッセージを必ずクリップボードチェインの次のウィンドウに送ります。次のウィンドウのハンドルの初期値は，***SetClipboardViewer*** 関数でウィンドウをチェインに追加したときの戻り値です。このハンドルの新しい値は ***WM_CHANGECBCHAIN*** メッセージで送られます。***WM_DESTROY*** メッセージを受信したウィンドウはクリップボードチェインから外れなければなりません。

WM_DRAWITEM

オーナー描画ボタン，コンボボックス，リストボックス，メニューに対し再描画を通知します。

引数 *wParam*

ありません。

lParam

項目情報と描画操作情報を含む ***DRAWITEMSTRUCT*** 構造体へのポインタ。

戻り値 ありません。

コメント ***DRAWITEMSTRUCT*** 構造体にあるディスプレイコンテキストに対して選択したオブジェクトがあれば，このメッセージを終了する前に残らず復元します。

WM_ENABLE

ウィンドウに対し使用許可または使用禁止を通知します。

引数 *wParam*

0の場合にウィンドウは使用許可状態で，0以外の場合は使用禁止状態です。

lParam

ありません。

戻り値 ありません。

WM_ENDSESSION

WM_QUERYENDSESSION メッセージに対する応答が0以外のアプリケーションに，セッションが終了するかどうかを知らせます。

引数 *wParam*

0の場合，セッションは終了しません。0以外の場合はセッションが終了します。

lParam

ありません。

戻り値 ありません。

コメント *wParam* が0以外の場合，アプリケーションは ***WM_ENDSESSION*** メッセージの処理中に，終了に必要な全タスクを実行するものとします。すべてのアプリケーションがこのメッセージ処理を完了した後で，Windowsが終了します。***WM_DESTROY*** メッセージの場合のように，***DestroyWindow*** や ***PostQuitMessage*** 関数を呼び出す必要はありません。

WM_ENTERIDLE

システムがアイドルになったのがモードつきダイアログの表示のためか，メニューの表示のためかをメインウィンドウ関数に通知します。

引数 *wParam*

システムのアイドル状態がダイアログボックス表示による場合は ***MSGF_DIALOGBOX***，メニュー表示による場合は ***MSGF_MENU***。

LOWORD *(lParam)*

wParam が ***MSGF_DIALOGBOX*** の場合はダイアログボックスへのハンドル，***MSGF_MENU*** の場合はメニューへのハンドル。

HIWORD *(lParam)*

ありません。

戻り値 ありません。

コメント キュー内のメッセージをすべて処理し，モードつきダイアログボックスまたは表示メニューのキューにメッセージがなくなると，システムはアイドルになります。***DefWindowProc*** が返すデフォルト値は 0 です。

WM_ERASEBKGND

描画のための無効領域の準備として，バックグラウンドの消去が必要であることをウィンドウに通知します。

引数 *wParam*

デバイスコンテキストへのハンドル。

lParam

ありません。

戻り値 アプリケーションがメッセージを処理してバックグラウンドを消去すれば 0 以外，その他の場合は 0。

コメント ***DefWindowProc*** 関数のデフォルト動作では，クラス構造体のクラスバックグラウンドブラシでバックグラウンドを消去します。クラスバックグラウンドブラシが 0 であれば，***UnrealizeObject*** 関数を呼び出して，ブラシ原点をウィンドウに位置合わせし，ブラシを選択してから，そのブラシでバックグラウンドの消去を行ないます。このときの Windows は，***MM_TEXT*** マッピングモードになります。デバイスコンテキストの使用モードが異なる場合は，クライアント領域の表示されている部分以外の領域が消去されることがあります。

WM_FONTCHANGE

フォントリソースのプールに変更があったことを最上位ウィンドウに知らせます。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント システムのフォントを追加または除去するアプリケーションは，***SendMessage*** 関数でウィンドウに対し必ずこのメッセージを送ってください。システムにフォントを追加するときは ***AddFontResource*** 関数を使います。フォントを除去するときは ***RemoveFontResource*** 関数を使います。

WM_GETDLGCODE

コントロールの方向キーと ***TAB*** キーの処理を，アプリケーション側でオーバーライドできるようにします。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 アプリケーション側で処理したい入力の種類に応じて ***DLGC_***定数の OR をとった値を返します。第 7 章『**Windows 定数**』の「DLGC_**ダイアログコード**」も参照してください。

コメント ***DefWindowProc*** が返すデフォルト値は 0 です。定義済みのコントロールクラスのウィンドウ関数から返されるコードは 0 以外の場合があります。このメッセージとデフォルト以外の戻り値が有効なのは，ユーザ定義のダイアログコントロールまたはサブクラス標準コントロールに限られます。

WM_GETFONT

ダイアログボックスの現在のフォントを返します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 システムフォントを使用するダイアログボックスの場合は0, その他の場合は使用フォントへのハンドルです。***WM_SETFONT*** も参照してください。

WM_GETMINMAXINFO

ウィンドウを最大化したときのデフォルトサイズ，最大化したときのデフォルト位置，トラッキングの最小と最大それぞれのデフォルトサイズの変更をウィンドウで行なえます。

引数 *wParam*
ありません。

lParam
5つの ***POINT*** からなる構造体の配列へのポインタ。*lParam* [0] は，Windows が内部的に使用します。*lParam* [1] は，最大化時のサイズ，*lParam* [2] は，最大化時のウィンドウの左上隅の位置，*lParam* [3] は，ウィンドウの最小トラッキングサイズです。*lParam* [4] は，最大トラッキングサイズです。

戻り値 *lParam* 配列要素の1から4は適宜変更可能です。

コメント トラッキングサイズとは，ウィンドウサイズの変更許容範囲を示すもので，最小と最大のサイズが規定されます。このメッセージで，Windows が使用するアプリケーションのデフォルトサイズを事前に変更できます。

WM_GETTEXT

ウィンドウに対応したテキストを受け取るバッファにコピーします。

引数 *wParam*

lParam にコピー可能な最大バイト数。

lParam

バッファへのポインタ。バッファの長さは *wParam* バイト長以上とします。

戻り値 ウィンドウが選択項目のないリストボックスの場合は ***LB_ERR***，エディットコントロールのないコンボボックスの場合は ***CB_ERR***，その他の場合は末尾のヌル文字を含む実際にコピーされるバイト数。

コメント エディットコントロールの内容，ボタンコントロールのボタン名，リストボックスの現在の選択項目，コンボボックスのエディットコントロール内容がそれぞれのテキストです。その他のウィンドウの場合は，ウィンドウタイトルがテキストです。***WM_GETTEXTLENGTH, WM_SETTEXT*** も参照してください。

WM_GETTEXTLENGTH

ウィンドウに対応したテキストのバイト長を返します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 対応テキストの長さ（末尾のヌル文字以外）が返されます。

コメント エディットコントロールの内容，ボタンコントロールのボタン名，リストボックスの現在の選択項目，コンボボックスのエディットコントロール内容がそれぞれのテキストです。その他のウィンドウの場合は，ウィンドウタイトルがテキストです。***WM_GETTEXT*** も参照してください。

WM_HSCROLL

水平スクロールバーのクリックをウィンドウに知らせます。

引数 *wParam*

スクロールバークリックの結果を記述するスクロールバーコード。水平スクロールバーコントロールに使用される ***SB_*** 定数のいずれかとします。第7章『**Windows 定数**』の「**SB_スクロールバーコマンド**」も参照してください。

***LOWORD** (lParam)*

ありません。

***HIWORD** (lParam)*

スクロールバーコントロールのハンドル。***WS_HSCROLL*** スタイルのウィンドウで作成されたコントロールの場合は 0。

戻り値 ありません。

コメント ウィンドウのテキストをスクロールするアプリケーションでは，***SetScrollPos*** 関数でスクロールボックスの位置をリセットする必要があります。

WM_HSCROLLCLIPBOARD

CF_OWNERDISPLAY 書式のクリップボードのオーナーに対し，クリップボードアプリケーションの水平スクロールバーがクリックされたことを知らせます。

引数 *wParam*

クリップボードアプリケーションウィンドウへのハンドル。

***LOWORD** (lParam)*

スクロールバークリックの結果を記述するスクロールバーコード。水平スクロールバーコントロールに使用される ***SB_*** 定数のいずれかとします。第7章『**Windows 定数**』の「**SB_スクロールバーコマンド**」も参照してください。

HIWORD *(lParam)*
ありません。

戻り値 ありません。

コメント クリップボードのオーナーは，クリップボードアプリケーションウィンドウを再描画するか，または ***InvalidateRect*** 関数を使用しなければなりません。クリップボードアプリケーションウィンドウのスクロールバー位置は，***SetScrollPos*** 関数でリセットが必要です。***CF_OWNERDISPLAY*** 書式の場合，クリップボードオーナーに送られるメッセージは，***WM_ASKCBFORMATNAME, WM_HSCROLLCLIPBOARD, WM_PAINTCLIPBOARD, WM_SIZECLIPBOARD, WM_VSCROLLCLIPBOARD*** です。クリップボードデータと書式の設定は ***SetClipboardData*** 関数で行ないます。

WM_ICONERASEBKGND

アイコンの描画準備としてバックグラウンドの塗りつぶしが必要なことを，最小化ウィンドウに知らせます。

引数 *wParam*
アイコンのデバイスコンテキスト。

lParam
ありません。

戻り値 ありません。

コメント このメッセージが有効なのは，クラスアイコンが定義されている最小化された，つまりアイコンのウィンドウだけです。他のウィンドウには ***WM_ERASEBKGND*** メッセージを送ります。***DefWindowProc*** 関数のデフォルト動作では，親ウィンドウのバックグラウンドブラシでアイコンのバックグラウンドが塗りつぶされます。

WM_INITDIALOG

表示されるダイアログボックスを初期化することをアプリケーションに指示します。

引数 *wParam*

入力フォーカスを持つダイアログボックス内の最初のコントロールID。

lParam

ダイアログを作成した関数に渡される *InitParam* 値。この引数を持つ関数は ***CreateDialogIndirectParam***, ***CreateDialogParam***, ***DialogBoxIndirectParam***, ***DialogBoxParam*** です。***CreateDialog***, ***CreateDialogIndirect***, ***DialogBox***, ***DialogBoxIndirect*** で作成されたダイアログの場合，この引数は0です。

戻り値 アプリケーションがダイアログの特定コントロールに入力フォーカスを設定する場合は 0，その他の場合は 0 以外。

コメント このメッセージによりアプリケーションはダイアログを初期化し，その表示直前に，ダイアログ内の特定コントロールに入力フォーカスを設定します。戻り値が0であれば，Windows によって *wParam* が示すコントロールに入力フォーカスが設定されます。*wParam* は通常，***WS_TABSTOP*** スタイルのダイアログボックスの最初のコントロールです。他の初期化方法としては，***WM_SETFONT*** メッセージによるダイアログコントロールのフォント設定を勧めます。

WM_INITMENU

メニューの表示をアプリケーションに予告します。

引数 *wParam*

メニューへのハンドル。

lParam

ありません。

戻り値 ありません。

コメント このメッセージが送られるのは，マウスによるメニューバーのクリック時またはメニューキーが押された時です。メニューの表示前に，メニュー項目の状態をアプリケーション側で変更できます。

WM_INITMENUPOPUP

ポップアップメニューの表示をアプリケーションに予告します。

引数 *wParam*

ポップアップメニューへのハンドル。

***LOWORD** (lParam)*

メインメニュー内のポップアップメニューのインデックス。

***HIWORD** (lParam)*

ポップアップメニューがシステムメニューの場合は 0 以外，その他の場合は 0。

戻り値 ありません。

コメント このメッセージにより，ポップアップメニューの表示前にアプリケーション側でメニュー項目を変更することができます。

W
Msg

WM_KEYDOWN

システムキー以外のキーが押されたことを，フォーカスのあるウィンドウに知らせます。

引数 *wParam*

仮想キーコード。

***LOWORD** (lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

***HIWORD** (lParam)*

ビット0から7まではOEM依存のキー走査コードです。拡張キーの場合はビット8が1です。**GRPH** の押下中にキーの押下が行なわれればビット13は1です。メッセージ転送前にキーが押下状態であればビット14は1です。ビット15はキーを離す場合は1に，押した場合には0になります。

戻り値 ありません。

コメント システムキー以外のキーとは，**GRPH** が押下状態でない時に押されたキーのことです。このメッセージの場合，***HIWORD*** *(lParam)* のビット13と15は0です。オートリピート機能のために，***WM_KEYUP*** メッセージの転送前に ***WM_KEYDOWN*** メッセージが複数回送られる場合があります。フォーカスのあるウィンドウがなければ，***WM_SYSKEYDOWN, WM_SYSCHAR, WM_SYSKEYUP*** の各メッセージが ***WM_KEYDOWN, WM_CHAR, WM_KEYUP*** のかわりに送られます。

WM_KEYUP

システムキー以外のキーが離されたことを，フォーカスのあるウィンドウに知らせます。

引数 *wParam*

仮想キーコード。

***LOWORD** (lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

***HIWORD** (lParam)*

ビット 0 から 7 までは OEM 依存のキー走査コードです。拡張キーの場合はビット 8 が 1 です。**GRPH** の押下中にキーの押下が行なわれればビット 13 は 1 です。メッセージ転送前にキーが押下状態であればビット 14 は 1 です。ビット 15 はキーを離す場合は 1 に，押す場合には 0 になります。

戻り値 ありません。

コメント システムキー以外のキーとは，**GRPH** キーが押下状態でない時に押されたキーのことです。このメッセージの場合，***HIWORD** (lParam)* のビット 13 は 0，ビット 15 は 1 です。オートリピート機能のために，***WM_KEYUP*** メッセージの転送前に ***WM_KEYDOWN*** メッセージが複数回送られる場合があります。フォーカスのあるウィンドウがなければ，***WM_SYSKEYDOWN, WM_SYSCHAR, WM_SYSKEYUP*** の各メッセージが ***WM_KEYDOWN, WM_CHAR, WM_KEYUP*** のかわりに送られます。

WM_KILLFOCUS

フォーカスの喪失をウィンドウに予告します。

引数 *wParam*

フォーカスを受け取るウィンドウのハンドル。

lParam

ありません。

戻り値 ありません。

コメント 表示中のキャレットがあればすべて破棄してください。

WM_LBUTTONDBLCLK

マウスの左ボタンがダブルクリックされたことをウィンドウに知らせます。

引数 *wParam*

押された仮想キーを示す値。***MK_***定数の組み合わせになります。第7章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

***LOWORD** (lParam)*

マウスカーソルのX座標。

***HIWORD** (lParam)*

マウスカーソルのY座標。

戻り値 ありません。

コメント このメッセージが有効なのは，***CS_DBLCLKS*** スタイルで作成されたウィンドウだけです。マウス座標の原点はウィンドウの左上隅です。ダブルクリックとは，システムのダブルクリック制限時間内に行なわれた2回の単独クリックのことです。ダブルクリックによりダウンクリックメッセージ，アップクリックメッセージ，ダブルクリックメッセージ，2回目のアップクリックメッセージが発生します。下記の関連項目も参照してください。

WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_LBUTTONDOWN

マウスの左ボタンがクリックされたことをウィンドウに知らせます。

引数 *wParam*

押された仮想キーを示す値。***MK_*** 定数の組み合わせになります。第7章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点はウィンドウの左上隅です。下記の関連項目も参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_LBUTTONUP

マウスの左ボタンが離されたことをウィンドウに知らせます。

引数 *wParam*

押された仮想キーを示す値です。***MK_*** 定数の組み合わせになります。第7章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点はウィンドウの左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_MBUTTONDBLCLK

マウスの中央ボタンがダブルクリックされたことをウィンドウに知らせます。

引数 *wParam*

押された仮想キーを示す値。***MK_*** 定数の組み合わせになります。第7章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

LOWORD *(lParam)*
マウスカーソルの X 座標。

HIWORD *(lParam)*
マウスカーソルの Y 座標。

戻り値 ありません。

コメント このメッセージが有効なのは ***CS_DBLCLKS*** スタイルで作成されたウィンドウだけです。マウス座標の原点はウィンドウの左上隅です。ダブルクリックとは，システムのダブルクリック制限時間内に行なわれた 2 回の単独クリックのことです。ダブルクリックによりダウンクリックメッセージ，アップクリックメッセージ，ダブルクリックメッセージ，2 回目のアップクリックメッセージが発生します。下記の関連項目も参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_MBUTTONDOWN

マウスの中央ボタンがクリックされたことをウィンドウに知らせます。

引数 *wParam*
押された仮想キーを示す値。***MK***_定数の組み合わせになります。第 7 章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

LOWORD *(lParam)*
マウスカーソルの X 座標。

HIWORD *(lParam)*
マウスカーソルの Y 座標。

戻り値　ありません。

コメント　マウス座標の原点はウィンドウの左上隅です。下記の関連項目も参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_MBUTTONUP

マウスの中央ボタンが離されたことをウィンドウに知らせます。

引数　*wParam*

押された仮想キーを示す値。***MK_***定数の組み合わせになります。第7章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

LOWORD *(lParam)*

マウスカーソルのX座標。

HIWORD *(lParam)*

マウスカーソルのY座標。

戻り値　ありません。

コメント　マウス座標の原点はウィンドウの左上隅です。下記の関連項目も参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_MDIACTIVATE

マルチドキュメントインターフェース（MDI）のクライアントウィンドウに対し，別の MDI 子ウィンドウをアクティブにするよう要求します。その後，このメッセージはアクティブ状態を解除される MDI 子ウィンドウとアクティブ化される MDI 子ウィンドウの両方に渡されます。

引数 *wParam*

MDI クライアントウィンドウには *wParam* を使用しません。子ウィンドウの場合，この引数が 0 であればアクティブ状態の解除，0 以外であればアクティブ化を示します。

LOWORD *(lParam)*

アクティブにする MDI 子ウィンドウへのハンドル。

HIWORD *(lParam)*

アクティブ状態を解除する MDI 子ウィンドウへのハンドル。

戻り値 ありません。

コメント このメッセージを受け取ったMDI クライアントウィンドウは，***WM_MDIACTIVATE*** メッセージを該当する *wParam* 引数とともに，非アクティブ化対象の MDI 子ウィンドウとアクティブ化対象の MDI 子ウィンドウの両方に送らなければなりません。MDI フレームウィンドウがアクティブになると，***WM_MDIACTIVATE*** メッセージを 0 以外の *wParam* で受け取った最後の MDI 子ウィンドウは，***WM_NCACTIVATE*** メッセージだけを受け取り，別の ***WM_MDIACTIVATE*** メッセージは受け取りません。非アクティブ化対象の MDI 子ウィンドウが最大化されている場合は復元されます。アクティブ化対象の MDI 子ウィンドウは最大化されます。

WM_MDICASCADE

マルチドキュメントインターフェース（MDI）のクライアントウィンドウのMDI 子ウィンドウをカスケード表示の形式に配置します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

WM_MDICREATE

マルチドキュメントインターフェース（MDI）の子ウィンドウを MDI クライアントウィンドウに作成します。

引数 *wParam*
ありません。

lParam
MDICREATE 構造体へのポインタ。

戻り値 戻り値の上位ワードは 0, 下位ワードは新しく作成された MDI 子ウィンドウの ID。

コメント MDI 子ウィンドウは次のスタイルで作成されます。***WS_CHILD, WS_CLIPSIBLINGS, WS_CLIPCHILDREN, WS_SYSMENU, WS_CAPTION, WS_THICKFRAME, WS_MINIMIZEBOX, WS_MAXIMIZEBOX*** の各スタイルと *lParam* の ***MDICREATESTRUCT*** 構造体にあるスタイルです。MDI 子ウィンドウのタイトルは MDI フレームウィンドウのウィンドウメニューに追加されます。クライアントウィンドウの子ウィンドウはすべてこのメッセージで作成してください。作成された MDI 子ウィンドウには ***WM_CREATE*** メッセージが *lParam* 引数とともに送られます。*lParam* 引数は ***CREATESTRUCT*** 構造体へのポインタです。この構造体には，MDI 子ウィンドウを作成した ***WM_MDICREATE*** メッセージに送られた

MDICREATESTRUCT 構造体を指すフィールドがあります。このメッセージは再入不能なので，たとえば MDI 子ウィンドウが ***WM_CREATE*** メッセージを処理中であると， ***WM_MDICREATE*** メッセージを送ることができません。

WM_MDIDESTROY

マルチドキュメントインターフェース（MDI）のクライアントウィンドウに，MDI 子ウィンドウのクローズを要求します。

引数 *wParam*
MDI 子ウィンドウへのハンドル。

lParam
ありません。

戻り値 ありません。

コメント MDI 子ウィンドウのタイトルがフレームウィンドウから除去され，MDI 子ウィンドウはアクティブ状態を解除されます。MDI 子ウィンドウの解除を行なう場合は，必ずこのメッセージで行ないます。

WM_MDIGETACTIVE

アクティブなマルチドキュメントインターフェース（MDI）の子ウィンドウと，それが最大化されているかどうかの情報を返します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 戻り値の下位ワードはアクティブな MDI 子ウィンドウへのハンドル。MDI 子ウィンドウが最大化されていれば，上位ワードは 1，最大化されていなければ 0。

WM_MDIICONARRANGE

マルチドキュメントインターフェース（MDI）のクライアントウィンドウで最小化されているMDI子ウィンドウを並べ換えます。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント アイコン化されていないMDI子ウィンドウは変更しません。

WM_MDIMAXIMIZE

マルチドキュメントインターフェース（MDI）のクライアントウィンドウにMDI子ウィンドウの最大化を要求します。

引数 *wParam*

MDI子ウィンドウのIDです。

lParam

ありません。

戻り値 ありません。

コメント 最大化されているMDI子ウィンドウのクライアントが，MDIクライアントウィンドウのクライアント領域と同じ大きさになります。最大化されたMDI子ウィンドウのシステムメニューは，MDIフレームウィンドウのメニューバーに配置されます。MDI子ウィンドウのタイトルは，MDIフレームウィンドウのタイトルに加えられます。

WM_MDINEXT

マルチドキュメントインターフェース（MDI）の次の子ウィンドウをアクティブにします。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント 次のMDI子ウィンドウは現在アクティブなMDI子ウィンドウの直後にあります。現在アクティブなMDI子ウィンドウの位置は，他のすべてのMDI子ウィンドウの後ろになります。

WM_MDIRESTORE

マルチドキュメントインターフェース（MDI）の最大化または最小化された子ウィンドウを復元します。

引数 *wParam*

MDI子ウィンドウのID。

lParam

ありません。

戻り値 ありません。

WM_MDISETMENU

マルチドキュメントインターフェース（MDI）のフレームウィンドウのメニューまたはウィンドウポップアップメニューを置き換えます。

引数 *wParam*

ありません。

LOWORD *(lParam)*

MDI フレームウィンドウの新しいメニューへのハンドル，または NULL。

HIWORD *(lParam)*

新しいウィンドウポップアップメニューへのハンドル，または NULL。

戻り値 置き換えられた MDI フレームウィンドウメニューのハンドル。

コメント ***LOWORD*** *(lParam)* または ***HIWORD*** *(lParam)* のいずれかが NULL であれば，対応するメニューに変更はありません。変更するには，このメッセージ転送後に ***DrawMenuBar*** 関数でメニューバーを更新します。前のウィンドウポップアップメニューの MDI 子ウィンドウメニュー項目が除去されて，新しいウィンドウポップアップメニューに配置されます。最大化されている MDI 子ウィンドウのシステムメニューと，復元されたコントロールが前の MDI フレームウィンドウメニューから除去され，新しい MDI フレームウィンドウメニューに加えられます。

WM_MDITILE

マルチドキュメントインターフェース（MDI）のクライアントウィンドウの MDI 子ウィンドウをタイル形式に配置します。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

WM_MEASUREITEM

オーナー描画ボタン，コンボボックス，リストボックス，メニューのオーナーに対しそれぞれのコントロールの寸法を問い合わせます。

引数 *wParam*

ありません。

lParam

MEASUREITEMSTRUCT 構造体へのポインタ。

戻り値 ありません。

コメント このメッセージが有効なのは次に限られます。***BS_OWNERDRAW*** スタイルのボタン，***CBS_OWNERDRAWFIXED*** または ***CBS_OWNERDRAWVARIABLE*** スタイルのコンボボックス，***LBS_OWNERDRAWFIXED*** または ***LBS_OWNERDRAWVARIABLE*** スタイルのリストボックス，オーナー描画メニュー項目です。このメッセージはコントロールの作成時コントロールのオーナーに送られます。*lParam* が指す ***MEASUREITEMSTRUCT*** 構造体には，このコントロールの正しい値を代入します。***LBS_OWNERDRAWVARIABLE*** スタイルのリストボックスと ***CBS_OWNERDRAWVARIABLE*** スタイルのコンボボックスには，このメッセージが各項目ごとに1度ずつ送られます。その他のスタイルでは，各コントロールまたはメニュー項目に1度ずつ送られます。***CBS_OWNERDRAWFIXED*** スタイルのコンボボックスまたは ***LBS_OWNERDRAWFIXED*** スタイルのリストボックスのあるダイアログの場合は，***WM_INITDIALOG*** メッセージに先立って ***WM_MEASUREITEM*** メッセージが送られます。

WM_MENUCHAR

メニューで未定義のニーモニック文字が押されたことを，現在のメニューのオーナーに知らせます。

引数 *wParam*

ニーモニック文字の ASCII 値。

***LOWORD** (lParam)*

ポップアップメニューの場合は ***MF_POPUP***，システムメニューの場合は ***MF_SYSMENU***。

***HIWORD** (lParam)*

現在のメニューへのハンドル。

戻り値 戻り値の上位ワードが 0 であれば，ニーモニック文字が廃棄され，スピーカにビープ音が送られます。上位ワードが 1 であれば，現在のメニューがクローズされます。上位ワードが 2 であれば，下位ワードのメニュー項目が選択されます。

コメント メニューのビットマップ選択にアクセラレータを使用するときは，アプリケーションがこのメッセージを処理しなければなりません。

WM_MENUSELECT

メニュー項目が選択されたことをメニューのオーナーに知らせます。

引数 *wParam*

メニュー項目 ID またはポップアップメニューのハンドル。

***LOWORD** (lParam)*

-1 または次の定数の組み合わせ（第 7 章『**Windows 定数**』の「**MF_メニューフラグ**」を参照)。

MF_BITMAP, MF_CHECKED, MF_DISABLED, MF_GRAYED, MF_MOUSESELECT, MF_OWNERDRAW, MF_POPUP, MF_SYSMENU

HIWORD *(lParam)*

メニューがシステムメニューまたは *lParam* が -1 の場合は 0, その他の場合はメニューへのハンドル。

戻り値 ありません。

コメント *lParamLo* が -1 で *lParamHi* が 0 の場合は，メニュー外でのマウスのクリックまたは ***ESC*** キーが押されたことにより，メニューがクローズされています。

WM_MOUSEACTIVATE

アクティブでないウィンドウに対しマウスがクリックされたことを通知します。

引数 *wParam*

ウィンドウの最上位の親ウィンドウへのハンドル。

LOWORD *(lParam)*

HT 定数のいずれか（第7章『**Windows 定数**』の「**HT ヒットテストコード**」を参照)。***WM_NCHITTEST*** メッセージが返す値と同じです。

HIWORD *(lParam)*

マウスメッセージ番号。

戻り値 このメッセージを最初に受け取るウィンドウは，戻り値が ***MA_ACTIVATE*** であればアクティブになります。***MA_NOACTIVATE*** であれば，アクティブになりません。***MA_ACTIVATEANDEAT*** であれば，ウィンドウがアクティブ化され，マウスイベントが廃棄されます。正当な戻り値は上記のみです。

コメント このメッセージを各ウィンドウから ***DefWindowProc*** に渡し，そこから親ウィンドウに渡すこともできます。0 以外の戻り値が発生すると，この処理はそこで停止してウィンドウはアクティブになりません。関連項目として ***WM_NCHITTEST*** も参照してください。

WM_MOUSEMOVE

クライアント領域でマウスの移動があったことをウィンドウに知らせます。

引数 *wParam*

押された仮想キーを示す値。***MK_***定数の組み合わせになります（第7章『**Windows 定数**』の「**MK_キー状態マスク**」を参照）。

LOWORD *(lParam)*

マウスカーソルのX座標。

HIWORD *(lParam)*

マウスカーソルのY座標。

戻り値 ありません。

コメント マウス座標の原点はウィンドウの左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_MOVE

ウィンドウが移動されたことを通知します。

引数 *wParam*

ありません。

LOWORD *(lParam)*

クライアント領域の左上隅の新しい X 座標。

HIWORD *(lParam)*

クライアント領域の左上隅の新しい Y 座標。

戻り値 ありません。

コメント ポップアップとオーバーラップウィンドウの新しい座標は，画面に相対的な位置になります。子ウィンドウの新しい座標は，親ウィンドウのクライアント領域に相対的な位置になります。

WM_NCACTIVATE

アクティブ状態または非アクティブ状態を示すために，タイトルバーまたはアイコンの変更が必要であることをウィンドウに通知します。

引数 *wParam*

0 でウィンドウの非アクティブ化，0 以外でアクティブ化。

lParam

ありません。

戻り値 ありません。

コメント ***DefWindowProc*** のデフォルト動作では，アクティブでないウィンドウには淡色のタイトル，アクティブなウィンドウには黒のタイトルが描画されます。この時点ではアクティブなアイコンとアクティブでないアイコンの識別は行なわれません。

WM_NCCALCSIZE

クライアント領域のサイズをウィンドウに問い合わせます。

引数 *wParam*

ありません。

lParam

ウィンドウの画面座標を含む ***RECT*** 構造体へのポインタ。

戻り値 ありません。

コメント *lParam* で指定する領域には，クライアント領域，境界線，タイトル，スクロールバーが含まれます。***DefWindowProc*** のデフォルト動作では，クライアント領域のサイズが計算されます。境界線，タイトル，スクロールバーも含めクライアント領域が計算され，計算結果は *lParam* の ***RECT*** 構造体に格納します。

WM_NCCREATE

ウィンドウの作成開始をアプリケーションに直前に予告します。

引数 *wParam*

作成するウィンドウへのハンドル。

lParam

ウィンドウの ***CREATESTRUCT*** 構造体へのポインタ。

戻り値 ウィンドウの作成が成功すれば 0 以外，失敗すれば 0。

コメント このメッセージの戻り値が 0 の場合は，***CreateWindow*** 関数または他のウィンドウ作成関数の戻り値も 0 になります。***DefWindowProc*** のデフォルト動作では，スクロールバーの初期化，ウィンドウテキストの設定，内部データ構造体へのメモリ割り当てが行なわれます。

WM_NCDESTROY

非クライアント領域の破棄をウィンドウに予告します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

コメント このメッセージを送るのは，***WM_DESTROY*** メッセージ転送後の ***Destroy-Window*** 関数です。***DefWindowProc*** のデフォルト動作では，このウィンドウに対応したメモリがすべて解放されます。

WM_NCHITTEST

マウス入力を受け取るウィンドウに，マウスが移動したことを知らせます。

引数 *wParam*
ありません。

LOWORD *(lParam)*
マウスカーソルの X 座標。

HIWORD *(lParam)*
マウスカーソルの Y 座標。

戻り値 このメッセージが ***DefWindowProc*** に渡される場合のデフォルトの戻り値は，***HT*** 定数のいずれか（第 7 章『**Windows 定数**』の「**HT ヒットテストコード**」を参照）。

HTBOTTOM, HTBOTTOMLEFT, HTBOTTOMRIGHT, HTCAPTION, HTCLIENT, HTERROR, HTGROWBOX, HTHSCROLL, HTLEFT, HTMENU, HTNOWHERE, HTREDUCE, HTRIGHT, HTSIZE, HTSYSMENU, HTTOP, HTTOPLEFT, HTTOPRIGHT, HTTRANSPARENT, HTVSCROLL, HTZOOM

コメント マウス座標の原点は画面の左上隅です。通常はマウスカーソルのあるウィンドウがすべてのマウス入力を受け取りますが，これを **GetCapture** 関数でオーバーライドすることもできます。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_NCLBUTTONDBLCLK

非クライアント領域でのマウスの左ボタンがダブルクリックされたことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第7章『**Windows 定数**』の「**HT ヒットテストコード**」を参照)。***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点は画面の左上隅です。ダブルクリックとは，システムのダブルクリック制限時間内に行なわれる2回の単独クリックのことです。ダブルクリックによりダウンクリックメッセージ，アップクリックメッセージ，ダブルクリックメッセージ，2回目のアップクリックメッセージが発生します。***DefWindowProc*** のデフォルト動作では，対応する非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_NCLBUTTONDOWN

非クライアント領域でマウスの左ボタンがクリックされたことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第7章『**Windows 定数**』の「**HT ヒットテストコード**」を参照）。***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点は画面の左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NCLBUTTONUP

非クライアント領域でマウスの左ボタンが離されたことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第7章『**Windows 定数**』の「**HT ヒットテストコード**」を参照）。***WM_NCHITTEST*** メッセージが返す値と同じです。

***LOWORD** (lParam)*

マウスカーソルの X 座標。

***HIWORD** (lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点は画面の左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP,

WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NCMBUTTONDBLCLK

非クライアント領域でマウスの中央ボタンがダブルクリックされたことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第 7 章『**Windows 定数**』の「**HT ヒットテストコード**」を参照)。 ***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点は画面の左上隅です。ダブルクリックとは，システムのダブルクリック制限時間内に行なわれる 2 回の単独クリックのことです。ダブルクリックによりダウンクリックメッセージ，アップクリックメッセージ，ダブルクリックメッセージ，2 回目のアップクリックメッセージが発生します。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NCMBUTTONDOWN

非クライアント領域でマウスの中央ボタンがクリックされたことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第 7 章『**Windows 定数**』の「**HT ヒットテストコード**」を参照)。 ***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点は画面の左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NCMBUTTONUP

非クライアント領域でマウスの中央ボタンが離されたことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第 7 章『**Windows 定数**』の「**HT ヒットテストコード**」を参照)。***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点は画面の左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NCMOUSEMOVE

非クライアント領域でマウスが移動したことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第 7 章『**Windows 定数**』の「**HT ヒットテストコード**」を参照）。***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点は画面の左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NCPAINT

フレームの描画が必要であることをウィンドウに知らせます。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント このメッセージに対するアプリケーションの応答は，次の2つの場合に分かれます。カスタムウィンドウフレームを描画する場合と ***DefWindowProc*** にメッセージを渡して，標準ウィンドウフレームを描画する場合です。カスタムフレームのクリッピング領域は，フレームの形にかかわらず，矩形になります。

WM_NCRBUTTONDBLCLK

非クライアント領域でマウスの右ボタンがダブルクリックされたことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第7章『**Windows 定数**』の「**HT ヒットテストコード**」を参照）。***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント　マウス座標の原点は画面の左上隅です。ダブルクリックとは，システムのダブルクリック制限時間内に行なわれる2回の単独クリックのことです。ダブルクリックによりダウンクリックメッセージ，アップクリックメッセージ，ダブルクリックメッセージ，2回目のアップクリックメッセージが発生します。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NCRBUTTONDOWN

非クライアント領域でマウスの右ボタンがクリックされたことをウィンドウに知らせます。

引数　*wParam*

HT 定数のいずれか（第7章『**Windows 定数**』の「**HT ヒットテストコード**」を参照）。***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD *(lParam)*

マウスカーソルの X 座標。

HIWORD *(lParam)*

マウスカーソルの Y 座標。

戻り値　ありません。

コメント　マウス座標の原点は画面の左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NCRBUTTONUP

非クライアント領域でマウスの右ボタンが離されたことをウィンドウに知らせます。

引数 *wParam*

HT 定数のいずれか（第7章『**Windows 定数**』の「**HT ヒットテストコード**」を参照)。***WM_NCHITTEST*** メッセージが返す値と同じです。

LOWORD (lParam)

マウスカーソルの X 座標です。

HIWORD (lParam)

マウスカーソルの Y 座標です。

戻り値 ありません。

コメント　マウス座標の原点は画面の左上隅です。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCHITTEST, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN, WM_RBUTTONUP

DefWindowProc のデフォルト動作では，非クライアント領域に該当する ***WM_SYSCOMMAND*** メッセージが送られます。

WM_NEXTDLGCTL

ダイアログボックスのコントロールフォーカスを変更します。

引数　*wParam*

lParam が 0 以外の場合は，*wParam* はフォーカスを受けるコントロールへのハンドル。*lParam* が 0 で *wParam* が 0 の場合は，次のタブストップスタイルつきコントロールがフォーカスを受けます。*wParam* が 0 以外の場合は，前のタブストップスタイルつきコントロールがフォーカスを受けます。

lParam

0 の場合は，タブストップスタイルつきコントロールを探す方向が *wParam* によって決定されます。0 以外の場合，*wParam* はフォーカスを受け取るコントロールへのハンドルです。

戻り値　ありません。

コメント　***SetFocus*** 関数とは異なり，このメッセージはデフォルトコントロールの境界を変更します。このメッセージを送るには ***PostMessage*** 関数を使います。***SendMessage*** は使用しません。コントロールフォーカス設定用のメッセージが他にもあり，それを同時処理する必要がある場合のみ ***SendMessage*** を使います。

WM_PAINT

クライアント領域全体または一部の再描画が必要であることをウィンドウに知らせます。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント このメッセージが送られるのは，***UpdateWindow*** 関数が呼び出された場合と，***DispatchMessage*** 関数が ***WM_PAINT*** メッセージを受けた場合です。関連項目として ***BeginPaint, EndPaint*** を参照してください。

WM_PAINTCLIPBOARD

クリップボードの内容の全部または一部の表示をクリップボードのオーナーに要求します。

引数 *wParam*

クリップボードアプリケーションウィンドウへのハンドル。

LOWORD *(lParam)*

PAINTSTRUCT 構造体を含むメモリへのハンドル。この構造体で，クリップボードアプリケーションウィンドウのクライアント領域の描画部分を定義します。

HIWORD *(lParam)*

ありません。

戻り値 ありません。

コメント *PAINTSTRUCT* データへのハンドルは *GlobalLock* 関数で必ずロックしてから使います。このメッセージから戻るときまたは制御を渡すときは，あらかじめ必ず *GlobalUnlock* 関数でロックを解除します。*PAINTSTRUCT* の描画寸法を最後の *WM_SIZECLIPBOARD* メッセージで得られた寸法と比較してください。クリップボード書式が *CF_OWNERDISPLAY* のときにクリップボードオーナーに送られるメッセージは，*WM_ASKCBFORMATNAME*, *WM_HSCROLLCLIPBOARD*，*WM_PAINTCLIPBOARD*，*WM_SIZECLIPBOARD*, *WM_VSCROLLCLIPBOARD* です。クリップボードデータと書式の設定は *SetClipboardData* 関数によって行ないます。

WM_PAINTICON

クラスアイコンを持つ最小化されたウィンドウに，アイコンの描画を指示します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

コメント クラスアイコンの定義がないウィンドウにはかわりに *WM_PAINT* メッセージを送ります。*DefWindowProc* のデフォルト動作では，クラスアイコンのアイコンが描画されます。

WM_PALETTECHANGED

システムパレットの変更を全ウィンドウに知らせます。

引数 *wParam*
システムパレット変更後のウィンドウのハンドルです。

lParam
ありません。

戻り値　ありません。

コメント　このメッセージが送られるのは，入力フォーカスのあるウィンドウが論理パレットを実現してシステムパレットを変更した場合です。このメッセージが *wParam* のハンドルを持たないウィンドウに送られると，パレットを実現してクライアント領域を更新する場合があります。

WM_PARENTNOTIFY

子ウィンドウの作成，破棄，マウスによるクリックをすべての上位ウィンドウに知らせます。

引数　*wParam*

このメッセージの転送理由に応じて次の定数を取ります。

WM_CREATE, WM_DESTROY, WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN

LOWORD *(lParam)*

wParam が ***WM_CREATE*** または ***WM_DESTROY*** であれば子ウィンドウのハンドル，その他の場合はマウスカーソルの X 座標。

HIWORD *(lParam)*

wParam が ***WM_CREATE*** または ***WM_DESTROY*** であれば子ウィンドウの ID，その他の場合はマウスカーソルの Y 座標。

戻り値　ありません。

コメント　このメッセージを送るのは，子ウィンドウを作成する ***CreateWindow*** または ***CreateWindowEx*** 関数が戻る直前，および子ウィンドウ破棄のアクションが取られる前です。このメッセージは子ウィンドウの上位ウィンドウすべてに送られます。子ウィンドウが ***WS_EX_NOPARENTNOTIFY*** 拡張ウィンドウスタイルの場合，このメッセージは送りません。ダイアログボックスの子ウィンドウが ***WS_EX_NOPARENTNOTIFY*** 拡張ウィンドウスタイルを持つのは，引数が子ウィンドウに該当する ***CreateWindowEx*** 関数で作成されない場合です。

WM_PASTE

ウィンドウの現在のカーソル位置にクリップボードの内容をコピーします。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント クリップボードの内容が ***CF_TEXT*** 書式でなければ，コピーは行なわれません。

WM_QUERYDRAGICON

デフォルトのアイコンカーソルでアイコンをドラッグするか，またはアイコンを置換するかを，クラス定義のアイコンがない最小化ウィンドウに問い合わせます。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 0 であればデフォルトのアイコンカーソルが使われます。0 以外の場合，戻り値の下位ワードは新しく使用するカーソルのハンドルになります。カーソルハンドルの場合はディスプレイドライバの解像度に適合するモノクロカーソルに限ります。***LoadCursor*** 関数で実行可能リソースからカーソルをロードできます。

WM_QUERYENDSESSION

各アプリケーションに対してセッションの終了確認をします。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 アプリケーションの終了が可能であれば 0 以外，終了できなければ 0。

コメント このメッセージは各アプリケーションに送られます。アプリケーションがすべて 0 以外を返すか，または 0 を返すアプリケーションが現われればメッセージ転送は終了します。0 を返すアプリケーションがあると，セッションは終了されません。さらに，その時点までの戻り値が 0 以外のアプリケーションには，*wParam* を 0 にした ***WM_ENDSESSION*** メッセージが送られます。***DefWindowProc*** のデフォルト動作では 0 以外が返されます。

WM_QUERYNEWPALETTE

入力フォーカスを受けたとき論理パレットを実現するかどうかをウィンドウに問い合わせます。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 入力フォーカスを受けて論理パレットを実現する場合は 0 以外，実現しない場合は 0。

WM_QUERYOPEN

アイコンを開いてウィンドウとして表示できるかどうかをアプリケーションに問い合わせます。

引数　*wParam*

ありません。

lParam

ありません。

戻り値　0であればアプリケーションはオープンされず，0以外であればオープンされます。***DefWindowProc***のデフォルト動作では，0以外の値が返されます。

WM_QUIT

実行終了をアプリケーションに指示します。

引数　*wParam*

PostQuitMessage関数呼び出しでWindowsに渡される終了コード。

lParam

ありません。

戻り値　ありません。

コメント　このメッセージ転送後の***GetMessage***関数呼び出しは0を返します。*wParam*の終了コードは必ず保管してプログラムの終了コードとして使わなければなりません。

WM_RBUTTONDBLCLK

マウスの右ボタンがダブルクリックされたことをウィンドウに知らせます。

引数 *wParam*

押された仮想キーを示す値。***MK_*** 定数の組み合わせになります。第7章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

***LOWORD** (lParam)*

マウスカーソルの X 座標。

***HIWORD** (lParam)*

マウスカーソルの Y 座標。

戻り値 ありません。

コメント このメッセージは ***CS_DBLCLKS*** スタイルで作成されたウィンドウだけを対象とします。マウス座標の原点は画面の左上隅です。ダブルクリックとは，システムのダブルクリック制限時間内に行なわれる2回の単独クリックのことです。ダブルクリックによりダウンクリックメッセージ，アップクリックメッセージ，ダブルクリックメッセージ，2回目のアップクリックメッセージが発生します。下記の関連項目を参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDOWN, WM_RBUTTONUP

WM_RBUTTONDOWN

マウスの右ボタンがクリックされたことをウィンドウに知らせます。

引数 *wParam*

押された仮想キーを示す値。***MK***_定数の組み合わせになります。第7章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

LOWORD *(lParam)*

マウスカーソルのX座標です。

HIWORD *(lParam)*

マウスカーソルのY座標です。

戻り値 ありません。

コメント マウス座標の原点はウィンドウの左上隅です。下記の関連項目も参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONUP

WM_RBUTTONUP

マウスの右ボタンが離されたことをウィンドウに知らせます。

引数 *wParam*

押された仮想キーを示す値。***MK***_定数の組み合わせになります。第7章『**Windows 定数**』の「**MK_キー状態マスク**」も参照してください。

LOWORD (lParam)
マウスカーソルの X 座標。

HIWORD (lParam)
マウスカーソルの Y 座標。

戻り値 ありません。

コメント マウス座標の原点はウィンドウの左上隅です。下記の関連項目も参照してください。

WM_LBUTTONDBLCLK, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_MBUTTONDBLCLK, WM_MBUTTONDOWN, WM_MBUTTONUP, WM_MOUSEMOVE, WM_NCLBUTTONDBLCLK, WM_NCLBUTTONDOWN, WM_NCLBUTTONUP, WM_NCMBUTTONDBLCLK, WM_NCMBUTTONDOWN, WM_NCMBUTTONUP, WM_NCMOUSEMOVE, WM_NCRBUTTONDBLCLK, WM_NCRBUTTONDOWN, WM_NCRBUTTONUP, WM_RBUTTONDBLCLK, WM_RBUTTONDOWN

WM_RENDERALLFORMATS

クリップボードデータをあらゆる既存書式で作成することを，クリップボードのオーナーに要求します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

コメント このメッセージがクリップボードオーナーのアプリケーションに送られるのは，そのアプリケーションを破棄するときです。各書式の生成，およびクリップボードへの書式化データのハンドル転送は，***SetClipboardData*** 関数によって行なってください。***WM_RENDERFORMAT*** も参照してください。

WM_RENDERFORMAT

要求書式のデータへのハンドルをクリップボードに送ることを，クリップボードのオーナーに要求します。

引数 *wParam*

SetClipboardData 関数に対する要求データ書式。

lParam

ありません。

戻り値 ありません。

コメント データの書式化は要求に応じて行なうものとします。書式化データへのハンドルは必ず ***SetClipboardData*** 関数でクリップボードに送ってください。***WM_RENDERALLFORMATS*** も参照してください。

WM_SETCURSOR

カーソルが移動したことを親ウィンドウに知らせます。

引数 *wParam*

カーソルのあるウィンドウへのハンドル。

LOWORD *(lParam)*

HT 定数のいずれか（第7章『**Windows 定数**』の「**HT ヒットテストコード**」を参照)。***WM_NCHITTEST*** メッセージが返す値と同じです。

HIWORD *(lParam)*

マウスメッセージ番号。

戻り値 0であれば ***DefWindowProc*** はデフォルト動作を続行し，0以外であればアクションは続行されません。

コメント ***DefWindowProc*** のデフォルト動作では，カーソルが非クライアント領域にあれば矢印になります。クライアント領域にあれば登録クラスのカーソルになりま

す。このメッセージによって，親ウィンドウはカーソルがどの子ウィンドウにあるかあるいはクライアント領域のどの部分にあるかによってカーソル形状を変更することができます。***SetCapture*** 関数によってマウス入力を捕える場合は，このメッセージは送られません。メッセージは元どおりの引数か，または変更して ***DefWindowProc*** に渡さなければなりません。***DefWindowProc*** にメッセージが渡り，***LOWORD*** *(lParam)* が ***HTERR*** で，***HIWORD*** *(lParam)* がマウスボタンの押下メッセージであれば，***MessageBeep*** 関数が呼び出されます。メニューモードのウィンドウの ***LOWORD*** *(lParam)* は 0 です。

WM_SETFOCUS

入力フォーカスの受け取りをウィンドウに通知します。

引数 *wParam*

入力フォーカスを喪失したウィンドウへのハンドル。

lParam

ありません。

戻り値 ありません。

コメント この時点でキャレットを表示する場合には，該当するキャレット表示用のキャレット関数を呼び出します。

WM_SETFONT

ダイアログボックスに使用するフォントを設定します。または，新たに作成されるコントロールのフォントを選択できることをダイアログボックスに知らせます。

引数 *wParam*

使用するフォントへのハンドル，またはシステムフォントを使うコントロールの場合は 0 です。

lParam

0 であればコントロールの再描画はありません。0 以外であれば再描画されます。

戻り値 ありません。

コメント ダイアログボックスの破棄などによりフォントが不要になったときは，***DeleteObject*** 関数でフォントを必ず削除します。ダイアログボックスの変更はフォント変更前に行なってください。このメッセージは，コントロール作成前の ***DS_SETFONT*** スタイルのダイアログボックスに送られます。***WM_INITDIALOG*** メッセージ受け取り時に，このメッセージを送ることもできます。

WM_SETREDRAW

ウィンドウの再描画フラグを設定または解除します。

引数 *wParam*

新しい再描画フラグ。0 であれば再描画は無効，0 以外であれば有効です。

lParam

ありません。

戻り値 ありません。

WM_SETTEXT

ウィンドウの対応テキストを設定します。

引数 *wParam*

ありません。

lParam

対応テキストとなるヌルで終わる文字列へのポインタ。

戻り値 対応テキストを設定するスペースが不充分であれば，リストボックスの場合は ***LB_ERRSPACE*** が返され，コンボボックスの場合は ***CB_ERRSPACE*** が返されます。項目が未選択のリストボックスの場合は，***LB_ERR*** が返されます。エディットコントロールのないコンボボックスの場合は，***CB_ERR*** が返されます。

コメント エディットコントロールの内容，ボタンコントロールのボタン名，リストボックスの現選択項目，コンボボックスのエディットコントロール内容がそれぞれのテキストです。その他のウィンドウの場合は，ウィンドウタイトルがテキストです。このメッセージでは，コンボボックスの現選択項目は変更されず，そのエディットコントロールの内容だけが変更されます。コンボボックスのエディットコントロール内のテキストに一致するコンボボックスのリストボックス項目を選択するには，***CB_SELECTSTRING*** を使います。***WM_GETTEXT*** も参照してください。

WM_SHOWWINDOW

ウィンドウの表示または非表示を予告します。

引数 *wParam*

0 でウィンドウの表示，0 以外で非表示。

lParam

このメッセージが ***ShowWindow*** 関数呼び出しによって送られる場合は 0。親ウィンドウがクローズするかポップアップウィンドウが非表示になる場合は ***SW_PARENTCLOSING***。親ウィンドウがオープンするかポップアップウィンドウが表示される場合は ***SW_PARENTOPENING***。

戻り値 ありません。

コメント ウィンドウの非表示または表示が行なわれるのは，***ShowWindow*** 関数が呼び出されたとき，オーバーラップウィンドウが最大化または復元されたとき，オーバーラップウィンドウまたはポップアップウィンドウがクローズまたはオープンされたときです。オーバーラップウィンドウがクローズされると，対応するポップアップウィンドウがすべて非表示になります。***DefWindowProc*** のデフォルト動作では，指示どおりにウィンドウが表示または非表示になります。

WM_SIZE

サイズ変更があったことをウィンドウに知らせます。

引数 *wParam*

Size 定数のいずれか（第 7 章**『Windows 定数』**の**「Size サイズ変更定数」**を参照)。

LOWORD (lParam)

クライアント領域の新しい幅。

HIWORD (lParam)

クライアント領域の新しい高さ。

戻り値 ありません。

コメント このメッセージでウィンドウの再描画を行なう場合は，***SetScrollPos*** 関数であれば *Redraw* 引数を 0 以外にします。***MoveWindow*** 関数であれば *Repaint* 引数を 0 以外にします。

WM_SIZECLIPBOARD

クリップボードアプリケーションウィンドウのサイズ変更があったことを，クリップボードのオーナーに知らせます。

引数 *wParam*

クリップボードアプリケーションウィンドウへのハンドル。

LOWORD *(lParam)*

クリップボードアプリケーションウィンドウの描画領域を指定する ***RECT*** 構造体。

HIWORD *(lParam)*

ありません。

戻り値 ありません。

コメント *wParam* が 0，すなわち ***RECT*** が（0，0，0，0）であれば，クリップボードアプリケーションは破棄されるか最小化され，クリップボードオーナーは表示リソースを解放できます。クリップボードアプリケーションウィンドウ描画用の ***PAINTSTRUCT*** 構造体は，***GlobalLock*** 関数でロックします。このメッセージから戻る場合は，事前に ***GlobalUnlock*** 関数でロック解除します。***LOWORD*** *(lParam)* の ***RECT*** をコピーして次の ***WM_PAINTCLIPBOARD*** メッセージに使います。クリップボード書式が ***CF_OWNERDISPLAY*** のときにクリップボードオーナーに送られるメッセージは，***WM_ASKCBFORMATNAME***，***WM_HSCROLLCLIPBOARD***，***WM_PAINTCLIPBOARD***，***WM_SIZECLIPBOARD***，***WM_VSCROLLCLIPBOARD*** です。クリップボードの書式設定は，***SetClipboardData*** 関数によるクリップボードのデータ設定と同時に行なわれます。

WM_SPOOLERSTATUS

プリントマネージャキューに対するジョブの追加または除去があったことをアプリケーションに知らせます。

引数 *wParam*

PR_JOBSTATUS にセット。

LOWORD *(lParam)*

プリントマネージャキューに現在あるジョブ数。

HIWORD *(lParam)*

ありません。

戻り値 ありません。

コメント 単に情報を伝えるメッセージで，プリントマネージャアプリケーションによって送られます。

WM_SYSCHAR

システムキーが押されたことをフォーカスがあるウィンドウに知らせます。どのウィンドウにも入力フォーカスがないときは，キーが押されたことをアクティブウィンドウに知らせます。

引数 *wParam*

キー値。

LOWORD *(lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

HIWORD *(lParam)*

ビット 0 から 7 までは OEM 依存のキー走査コードです。拡張キーの場合はビット 8 が 1 です。***GRPH*** キーの押下中にキーの押下が行なわれればビット 13 は 1 です。メッセージ転送前にキーが押下状態であればビット 14 は 1 です。ビット

15はキーを離す場合は1に，押す場合には0になります。

戻り値 ありません。

コメント このメッセージでは ***HIWORD*** *(lParam)*のビット15は0になります。システムキーが押されると *lParam* のビット13は1になり，どのウィンドウにも入力フォーカスがなければ0になります。*lParam* のビット13が0であれば，このメッセージは ***TranslateAccelerator*** 関数に渡されます。そうすれば入力フォーカスの有無に関わらずにアクティブウィンドウにアクセラレータを使用できます。どのウィンドウにも入力フォーカスがない場合は，***WM_SYSKEYDOWN, WM_SYSCHAR, WM_SYSKEYUP*** を ***WM_KEYDOWN, WM_CHAR, WM_KEYUP*** のかわりに送ります。関連項目として ***WM_CHAR, WM_DEADCHAR, WM_SYSDEADCHAR*** も参照してください。

WM_SYSCOLORCHANGE

システムカラー設定に変更があったことを最上位ウィンドウに知らせます。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

コメント システムカラー変更の影響を受けるウィンドウにはすべて ***WM_PAINT*** メッセージが送られます。アプリケーション側で既存のシステムカラーを使うブラシをすべて削除し，新しいシステムカラーを使うブラシに作り直す必要があります。

WM_SYSCOMMAND

システムメニュー項目，最小化ボックス，最大化ボックスが選択されたことをウィンドウに知らせます。

引数 *wParam*

システムコマンド要求で，***SC_*** 定数のいずれか（第 7 章『**Windows 定数**』の「**SC_システムコマンド値**」を参照）。Windows が内部的に使用する *wParam* の下位 4 ビットは *wParam* の使用前に 0 にセットします。

LOWORD *(lParam)*

マウスの X 座標。マウスが使用されなかった場合は 0。

HIWORD *(lParam)*

マウスの Y 座標。マウスが使用されなかった場合は 0。

戻り値 ありません。

コメント システムメニュー項目に割り当てたアクセラレータキーは，***WM_COMMAND*** メッセージではなくて ***WM_SYSCOMMAND*** メッセージに変換されます。***WM_COMMAND*** がアクセラレータに送られるのは，ウィンドウが最小化されていないときと，最小化されていてもアクセラレータに一致する項目がウィンドウメニューまたはシステムメニューにないときです。システムメニュー項目の変更を行なうには，***GetSystemMenu***，***AppendMenu***，***InsertMenu***，***ModifyMenu*** 関数を使用します。システムメニュー項目の選択を変更した場合，その処理はすべてアプリケーション側で行なわなければなりません。アプリケーションが処理しないメッセージは ***DefWindowProc*** に渡します。アプリケーションがシステムメニューコマンドを実行しなければならないときは ***WM_SYSCOMMAND*** メッセージを ***DefWindowProc*** に送ってかまいません。

WM_SYSDEADCHAR

デッドシステム文字をウィンドウに通知します。

引数 *wParam*

キー値。

***LOWORD** (lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

***HIWORD** (lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

戻り値 ありません。

コメント デッドキーには，ウムラウト文字とアクセントが含まれます。デッドシステムキーとは **GRPH** キーと組み合わせて使われるデッドキーのことです。このメッセージを使用して，キーを押しても単独では文字にならないキーのフィードバックにすることができます。関連項目として ***WM_CHAR***, ***WM_DEADCHAR***, ***WM_SYSCHAR*** も参照してください。

WM_SYSKEYDOWN

システムキーが押されたことをフォーカスのあるウィンドウに知らせます。入力フォーカスがどのウィンドウにもない場合は，アクティブウィンドウに知らせます。

引数 *wParam*

仮想キーコード。

***LOWORD** (lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

***HIWORD** (lParam)*

ビット 0 から 7 までは OEM 依存のキー走査コードです。拡張キーの場合はビット 8 が 1 です。**GRPH** キーの押下中にキーの押下が行なわれればビット 13 は 1 です。メッセージ転送前にキーが押下状態であればビット 14 は 1 です。ビット 15 はキーを離す場合は 1 に，押す場合には 0 になります。

戻り値 ありません。

コメント　このメッセージでは ***HIWORD*** *(lParam)* のビット 15 は 0 になります。システムキーが押されると *lParam* のビット 13 は 1 になり，どのウィンドウにも入力フォーカスがなければ 0 になります。*lParam* のビット 13 が 0 であれば，このメッセージは ***TranslateAccelerator*** 関数に渡されます。そうすれば入力フォーカスの有無に関わらずにアクティブウィンドウにアクセラレータを使用できます。オートリピート機能によって，複数の ***WM_SYSKEYDOWN*** メッセージが ***WM_SYSKEYUP*** メッセージの前に送られることがあります。どのウィンドウにも入力フォーカスがない場合は，***WM_SYSKEYDOWN, WM_SYSCHAR, WM_SYSKEYUP*** が ***WM_KEYDOWN, WM_CHAR, WM_KEYUP*** のかわりに送られます。

WM_SYSKEYUP

システムキーが離されたことをフォーカスのあるウィンドウに知らせます。入力フォーカスがどのウィンドウにもない場合は，アクティブウィンドウに知らせます。

引数　*wParam*

仮想キーコード。

LOWORD *(lParam)*

キーを押し続けたために繰り返されたキーストロークの回数。

HIWORD *(lParam)*

ビット 0 から 7 までは OEM 依存のキー走査コードです。拡張キーの場合はビット 8 が 1 です。***GRPH*** キーの押下中にキーの押下が行なわれればビット 13 は 1 です。メッセージ転送前にキーが押下状態であればビット 14 は 1 です。ビット 15 はキーを離す場合は 1 に，押す場合には 0 になります。

戻り値　ありません。

コメント　このメッセージでは ***HIWORD*** *(lParam)* のビット 15 は 1 になります。システムキーが離されると *lParam* のビット 13 は 1 になり，どのウィンドウにも入力フォーカスがなければ 0 になります。*lParam* のビット 13 が 0 であれば，このメッセージは ***TranslateAccelerator*** 関数に渡されます。そうすれば入力フォーカスの有無に関わらずにアクティブウィンドウにアクセラレータを使用できま

す。オートリピート機能によって，複数の ***WM_SYSKEYDOWN*** メッセージが ***WM_SYSKEYUP*** メッセージの前に送られることがあります。どのウィンドウにも入力フォーカスがない場合は，***WM_SYSKEYDOWN, WM_SYSCHAR, WM_SYSKEYUP*** が ***WM_KEYDOWN, WM_CHAR, WM_KEYUP*** のかわりに送られます。

WM_TIMECHANGE

システム時間に変更があったことを最上位ウィンドウに知らせます。

引数 *wParam*

ありません。

lParam

ありません。

戻り値 ありません。

コメント アプリケーションでシステム時間を変更したら，***SendMessage*** 関数によってすべての最上位ウィンドウにこのメッセージを送ってください。

WM_TIMER

タイマーの制限時間に達したことをアプリケーションに知らせます。

引数 *wParam*

タイマー ID。

lParam

タイマー作成時に ***SetTimer*** 関数に渡されたタイマー関数が 0 以外の場合のみ使用します。その場合のメッセージはメッセージキューに置かれず，タイマー関数に直接渡されます。

戻り値 ありません。

WM_UNDO

エディットコントロールの最後の操作を取り消します。

引数 *wParam*
ありません。

lParam
ありません。

戻り値 ありません。

WM_VKEYTOITEM

WM_KEYDOWN メッセージに対するリストボックスの反応を，リストボックスのオーナーに問い合わせます。

引数 *wParam*
押されたキー。

LOWORD *(lParam)*
リストボックスへのハンドル。

HIWORD *(lParam)*
現在のキャレット位置。

戻り値 -2の場合は，すべての処理がアプリケーションによって行なわれます。-1であれば，リストボックスがキーのデフォルト動作を実行します。0以上であれば，戻り値が示す項目に対して，リストボックスがキーのデフォルト動作を実行します。このメッセージは，***LBS_WANTKEYBOARDINPUT*** スタイルのリストボックスだけを対象とします。関連項目として ***WM_CHARTOITEM*** も参照してください。

WM_VSCROLL

垂直スクロールバーがクリックされたことをウィンドウに知らせます。

引数 *wParam*

スクロールバーのクリック結果を記述するスクロールコードで，垂直スクロールバーコントロールに該当する ***SB***_定数のいずれか（第7章『**Windows 定数**』の「**SB_スクロールバーコマンド**」を参照)。

***LOWORD** (lParam)*

ありません。

***HIWORD** (lParam)*

スクロールバーコントロールのハンドル。***WS_VSCROLL*** スタイルのウィンドウで作成されたコントロールの場合は 0。

戻り値 ありません。

コメント アプリケーションがウィンドウのテキストをスクロールする場合は，***SetScrollPos*** 関数でスクロールボックスの位置もリセットしなければなりません。

W
Msg

WM_VSCROLLCLIPBOARD

クリップボードアプリケーションで垂直スクロールバーがクリックされたことを，***CF_OWNERDISPLAY*** 書式のクリップボードのオーナーに知らせます。

引数 *wParam*

クリップボードアプリケーションウィンドウへのハンドル。

LOWORD *(lParam)*

スクロールバーのクリック結果を記述するスクロールバーコードで，垂直スクロールバーコントロールに使われる ***SB***_定数のいずれか（第7章『**Windows 定数**』の「**SB_スクロールバーコマンド**」を参照）。

HIWORD *(lParam)*

ありません。

戻り値 ありません。

コメント クリップボードオーナーはクリップボードアプリケーションウィンドウの再描画か，***InvalidateRect*** 関数の使用が必要です。クリップボードアプリケーションウィンドウのスクロールバー位置を，***SetScrollPos*** 関数で再設定する必要があります。クリップボード書式が ***CF_OWNERDISPLAY*** のときにクリップボードオーナーに送られるメッセージは，***WM_ASKCBFORMATNAME, WM_HSCROLLCLIPBOARD, WM_PAINTCLIPBOARD, WM_SIZECLIPBOARD, WM_VSCROLLCLIPBOARD*** です。クリップボードのデータと書式の設定は，***SetClipboardData*** 関数で行なわれます。

WM_WININICHANGE

Windows の初期化ファイルである WIN.INI に変更があったことを最上位ウィンドウに知らせます。

引数 *wParam*

ありません。

lParam

セクション名の文字列へのポインタ。

戻り値 ありません。*lParam* は NULL 以外とします。

コメント アプリケーションが Windows 初期化ファイルを変更するたびに，***SendMessage*** 関数で最上位ウィンドウすべてにこのメッセージを送る必要があります。

第6章

Windows型

Windowsでは多数の型が定義されています。型にはデータやハンドルばかりでなく，もっと複雑なものもあります。この章では，Windows型をアルファベット順に解説し，そのヘッダファイルを示します。**表6.1**は，型別のグループにまとめた一覧です。

➡ この一覧表には含めていませんが，他にnearポインタ型とfarポインタ型という相互に対応する型も定義されています。たとえば，***PBOOL***と***LPBOOL***はそれぞれ論理値を指すnearポインタとfarポインタです。

表6.1
この章で定義する型グループ

型	名前
データ	*BOOL, BYTE, DWORD, LPSTR, WORD*
ハンドル	*GLOBALHANDLE, HBITMAP, HBRUSH, HCURSOR, HDC, HFONT, HICON, HMENU, HPALETTE, HPEN, HRGN, HSTR, HWND*
標準	*DDEAck, DDEAdvise, DDEData, DDEPoke*
その他	上記以外の型

ATOM 型 WINDOWS.H

定義 typedef WORD ATOM ;

DDE 対応アプリケーション間の転送メッセージであるアトムを識別する 16 ビット数を定義します。

BITMAP 型 WINDOWS.H

定義
```
typedef struct tagBITMAP {
    int bmType ;
    int bmWidth ;
    int bmHeight ;
    int bmWidthBytes ;
    BYTE bmPlanes ;
    BYTE bmBitsPixel ;
    LPSTR bmBits ;
} BITMAP ;
```

BITMAP 構造体を使用するのは ***CreateBitmapIndirect*** と ***GetObject*** の両関数で，ビットマップのサイズ，色，ビット値を記述します。

フィールド ***bmType*** メンバはビットマップの種類を定義します。0 値は論理ビットマップです。***bmWidth*** はピクセル数による幅，***bmHeight*** はラスタライン数による高さです。ともに 0 より大きい値とします。***bmWidthBytes*** は各ラスタラインのバイト数です。偶数値とします。

bmPlanes はカラープレーンの数，***bmBitsPixel*** は各プレーン上の隣接するカラービットの数です。

bmBits はビットマップを実際に構成するビットのポインタです。ビットはバイトの配列という形をとります。

BITMAPCOREHEADER 型　　WINDOWS.H

定義
```
typedef struct tagBITMAPCOREHEADER {
    DWORD bcSize ;
    WORD bcWidth ;
    WORD bcHeight ;
    WORD bcPlanes ;
    WORD bcBitCount ;
} BITMAPCOREHEADER ;
```

デバイス独立のビットマップのサイズと色を定義します。***BITMAPCOREHEADER*** 構造体はデバイス独立のビットマップを完全に定義する ***BITMAPCOREINFO*** 構造体の一部として使われます。

フィールド

bcSize は ***BITMAPCOREHEADER*** 構造体のバイト数です。
bcWidth と ***bcHeight*** はピクセル数によるビットマップの幅と高さです。
bcPlanes は対象デバイスのプレーン数で，必ず 1 にセットします。
bcBitCount はピクセル当たりのビット数で，取りうる値は 1, 4, 8, 24 です。

コメント

bcBitCount の意味は以下のとおりです。

- ***bcBitCount*** が 1 の場合，ビットマップはモノクロで，カラーテーブルのエントリ数は 2 です。各ビットが 1 ピクセルを表わします。色分けは，ビットの 0 と 1 で行ないます。
- ***bcBitCount*** が 4 の場合，ビットマップは 16 色となり，0 から 15 までの番号が当てられます。各ピクセルの色を示すには 4 ビットが必要です。カラーテーブルのエントリ数は 16 となります。ビットマップの各バイトで 2 ピクセルを表わします。上位 4 ビットで 1 ピクセル，下位 4 ビットでもう 1 ピクセルです。
- ***bcBitCount*** が 8 の場合，ビットマップは 256 色となります。各ピクセルに 1 バイトが必要です。各バイトには 0 から 255 までのカラーテーブルへのインデックスが入ります。
- ***bcBitCount*** が 24 の場合，ビットマップは 224 色までとなります。カラーテーブルはありません。各ピクセルは 3 バイトからなり，それぞれ赤，緑，青の 3 色の輝度を示します。

BITMAPCOREINFO 型　　WINDOWS.H

定義

```
typedef struct tagBITMAPCOREINFO {
    BITMAPCOREHEADER bmciHeader;
    RGBTRIPLE bmciColors [1];
} BITMAPCOREINFO;
```

カラー情報テーブルと ***BITMAPCOREHEADER*** 構造体のサイズおよびカラー情報を組み合わせて，デバイス独立のビットマップを完全に定義します。

フィールド

bmciHeader は ***BITMAPCOREHEADER*** の構造体で，ビットマップのサイズとカラー情報を定義します。***bmciHeader*** の ***bcSize*** メンバは ***bmciColors*** メンバのオフセットとなります。

bmciColors は ***RGBTRIPLE*** 構造体の配列です。配列の要素数を決定するのは ***bmciHeader*** の ***bcBitCount*** です。***bmciColors*** は RGB カラー構造体の配列，または現在実現されている論理パレットに対するインデックスの配列です。どちらとして解釈するかは，デバイス独立ビットマップを呼び出す関数に渡される引数 *Usage* によって決まります。詳細については第 7 章の「**DIB_カラーテーブル識別子**」を参照してください。

コメント

ビットマップを使用するアプリケーションが 2 つ以上ある場合は，***bmciColors*** メンバに RGB 値を使われることを勧めます。ファイルに格納されるビットマップと別プログラムに転送されるビットマップには，パレットインデックスを使用しないでください。

BITMAPFILEHEADER 型　　WINDOWS.H

定義

```
typedef struct tagBITMAPFILEHEADER {
    WORD bfType;
    DWORD bfSize;
    WORD bfReserved 1;
    WORD bfReserved 2;
    DWORD bfOffBits;
```

} BITMAPFILEHEADER ;

デバイス独立のビットマップファイルのヘッダを定義します。ヘッダ内のデータでビットマップファイルの型，サイズとレイアウトが定義されます。

デバイス独立のビットマップファイルを構成するのは，順に ***BITMAPFILEHEADER***，***BITMAPINFO*** か ***BITMAPCOREINFO*** 構造体，実際のビットマップデータです。

フィールド ビットマップファイルヘッダ内の各メンバは次のように定義されます。

bfType メンバは，ファイルの型である BM を示します。

bfSize メンバは，ファイルサイズを 4 バイト長ブロックの数で示します。

bfReserved 1 と ***bfReserved 2*** は，文字どおり Windows の予約メンバです。

bfOffBits メンバは，ファイル内のビットマップ情報の実際の開始位置をバイト数で知らせます。

BITMAPINFO 型 WINDOWS.H

定義

```
typedef struct tagBITMAPINFO {
    BITMAPINFOHEADER bmiHeader ;
    RGBQUAD bmiColors [1] ;
} BITMAPINFO ;
```

Windows 3.0 のデバイス独立ビットマップのサイズとカラー情報が含まれます。実際のビットマップは，ピクセルを表わすバイトの配列で表現されます。

フィールド ***bmiHeader*** メンバには，ビットマップのサイズとカラー書式を定義する ***BITMAPINFOHEADER*** 構造体が含まれます。***bmiColors*** メンバは ***RGBQUAD*** 構造体の配列で，ビットマップの色を定義します。配列のエントリ数は ***bmiHeader*** 構造体の ***biBitCount*** メンバの値で決まります。

定義

```
typedef struct tagBITMAPINFOHEADER {
    DWORD biSize;
    DWORD biWidth;
    DWORD biHeight;
    WORD biPlanes;
    WORD biBitCount;
    DWORD bitCompression;
    DWORD biSizeImage;
    DWORD biXPelsPerMeter;
    DWORD biYPelsPerMeter;
    DWORD biClrUsed;
    DWORD biClrImportant;
} BITMAPINFOHEADER;
```

BITMAPINFO 構造体に使用されます。Windows 3.0 のデバイス独立ビットマップの寸法とカラー書式を定義します。

フィールド　***biSize*** メンバは構造体のサイズをバイト数で示します。***biWidth*** と ***biHeight*** はピクセル数によるビットマップの幅と高さです。***biPlane*** は対象デバイスのプレーン数で，1 にセットします。***biBitCount*** はビットマップの各ピクセルの記述に要するビット数を示します。***biCompression*** はビットマップに使用する圧縮の種別で，第 7 章で定義した ***BI***_定数のいずれかとします。***biSizeImage*** はビットマップイメージのバイト数によるサイズを示します。***biXPelsPerMeter*** と ***biYPelsPerMeter*** は対象デバイスの水平と垂直の解像度を示します。

ビットマップに実際に使用されるカラーテーブルのエントリ数は ***biClrUsed*** メンバで指定します。***biBitCount*** 値はエントリの最大数を示します。***biClrUsed*** を 0 とすると，最大数となります。1 から 23 は実際にアクセスされるカラー数を示します。24 は，***biClrUsed*** に示される参照カラーテーブルのサイズで，Windows によるカラーパレット性能の最適化に使われます。

biClrImportant メンバはビットマップの表示に必要と考えられるカラー数を示します。値を 0 とすると，色すべてが必要であることを示します。

コメント　***biBitCount*** の値の意味は，***BITMAPCOREHEADER*** 構造体の ***bcBitCount*** メンバの定義どおりです。

BOOL 型　　WINDOWS.H

定義　typedef int BOOL ;

16 ビット論理値。

BYTE 型　　WINDOWS.H

定義　typedef unsigned char BYTE ;

符号なしの 8 ビット値。

CATCHBUF 型　　WINDOWS.H

定義　typedef int CATCHBUF [9] ;

システムレジスタすべての状態を退避する構造体です。***Catch*** 関数と ***Throw*** 関数で使われます。

CLIENTCREATESTRUCT 型　　WINDOWS.H

定義
```
typedef struct tagCLIENTCREATESTRUCT {
    HANDLE hWindowMenu ;
    WORD idFirstChild ;
} CLIENTCREATESTRUCT ;
```

MDI クライアントウィンドウ作成時にウィンドウ ID とメニュー情報を格納します。

フィールド　***hWindowMenu*** メンバはアプリケーションのメニューへのハンドルです。***idFirstChild*** は MDI アプリケーションの最初の子ウィンドウの ID です。これにもとづいて，Windows がそれぞれの子ウィンドウの ID を割り当てます。

COLORREF 型　　WINDOWS.H

定義
```
typedef DWORD COLORREF ;
```

カラーに対応する 32 ビット値で，多くの GDI 関数に使用されます。上位ワードの上位バイトの値により 3 通りの形式を持ちます。

- 最上位バイトが 0 であれば，続く 3 バイトはそれぞれ青，緑，赤の RGB カラー輝度を表わします。0x00FF0000 は輝度 100% の青，0x0000FF00 は緑，0x000000FF は赤です。0x00000000 は黒で，0x00FFFFFF は白です。RGB カラー値は，***RGB*** 関数で簡単に ***COLORREF*** 値に変換できます。
- 最上位のバイトが 1 であれば，次に続くバイトは 0 にします。下位ワード（次の 2 バイト）が論理パレットに対するインデックスになります。したがって，0x01000000 はパレットの最初のエントリとなりインデックス 0 を示します。整数型のパレットインデックスは，***PaletteIndex*** 関数で ***COLORREF*** 値に変換できます。
- 最上位バイトが 2 の場合は，0 の場合と同様に，続く 3 バイトが RGB カラー輝度を表わします。ただし，値に当てられる色は，現在のデバイスコンテキストの論理パレットの中の最も近い色になります。パレットに対応した RGB の ***COLORREF*** 値は，***PaletteRGB*** 関数で RGB 値から生成されます。

コメント　パレットインデックス値とパレット対応 ***COLORREF*** 値をデバイスコンテキストに合わせるには次の方法を取ります。まず書き込み対象となるデバイスコンテキストにアプリケーションのパレットを選択します。選択は ***SelectPalette*** で行ないます。次に ***RealizePalette*** で実現します。これで描画関数に必要なパレットの色が用意されます。論理描画ツールを作成する場合も，同様にして事前にパレットの選択と実現を行ない，必要なカラーの生成を行ないます。

COMPAREITEMSTRUCT 型　　WINDOWS.H

定義

```
typedef struct tagCOMPAREITEMSTRUCT {
    WORD CtlType;
    WORD CtlID;
    HWND hwndItem;
    WORD itemID 1;
    DWORD itemData 1;
    WORD itemID 2;
    DWORD itemData 2;
} COMPAREITEMSTRUCT;
```

ソートされたオーナー描画コンボボックスまたはリストボックスの項目を比較する構造体です。コントロールへの項目追加によって ***WM_COMPAREITEM*** メッセージが送られてきます。このメッセージのパラメータの１つが，この型へのポインタです。メッセージを受け取り次第，オーナーは構造体の項目比較を行ない，結果の値を返します。詳細については第５章の「**WM_COMPAREITEM メッセージ**」を参照してください。

フィールド　***CtlType*** メンバに含まれる ***ODT_*** 定数は，***ODT_COMBOBOX*** であればオーナー描画コンボボックスを，***ODT_LISTBOX*** であればリストボックスを示します。***CtlID*** はコントロール ID で ***hwndItem*** はコントロールのウィンドウハンドルです。

itemID 1 は比較対象となるリストボックスまたはコンボボックスの第１項目のインデックスです。***itemData 1*** はその項目のデータです。***itemData 1*** のデータは，項目を追加したメッセージの *lParam* パラメータとして渡されます。第２項目に対しては ***itemID 2*** と ***itemData 2*** を使用します。

COMSTAT 型　　WINDOWS.H

定義
```
typedef struct tagCOMSTAT {
    BYTE fCtsHold : 1 ;
    BYTE fDsrHold : 1 ;
    BYTE fRlsdHold : 1 ;
    BYTE fXoffHold : 1 ;
    BYTE fXoffSent : 1 ;
    BYTE fEof : 1 ;
    BYTE fTxim : 1 ;
    WORD cbInQue ;
    WORD cbOutQue ;
} COMSTAT ;
```

通信デバイスのステータス情報を格納する構造体です。***GetCommError*** 関数で使います。

フィールド　各メンバの意味を以下に示します。

fCtsHold	送信可能待ち。
fDsrHold	***DSR*** 信号待ち。
fRlsdHold	***RLSD*** 信号待ち。
fXoffHold	***XoffChar*** 文字の受信による待ち。
fXoffSent	***XoffChar*** 文字の送信による待ち。
fEof	***EofChar*** 文字が受信されたかどうかを示す。
fTxim	送信待ちの文字があるかどうかを示す。

cbInQue で受信キュー内の文字数, ***cbOutQue*** で送信キュー内の文字数を示します。

CREATESTRUCT 型　　WINDOWS.H

定義
```
typedef struct tagCREATESTRUCT {
    LPSTR lpCreateParams ;
    HANDLE hInstance ;
```

```
    HANDLE hMenu ;
    HWND hwndParent ;
    int cy ;
    int cx ;
    int y ;
    int x ;
    LONG style ;
    LPSTR lpszName ;
    LPSTR lpszClass ;
    DWORD dwExStyle ;
} CREATESTRUCT ;
```

アプリケーションのウィンドウ関数に初期化パラメータを渡す構造体です。

フィールド ***lpCreateParams*** はウィンドウ作成データへのポインタです。***hInstance, hMenu, hwndParent*** はウィンドウ，メニュー，親ウィンドウを所有するモジュールのモジュールインスタンスへのハンドルです。作成対象のウィンドウがアプリケーションのメインウィンドウであれば，***hwndParent*** は NULL となります。

cx はウィンドウの幅，***cy*** は高さです。***x*** と ***y*** はウィンドウの左上隅の水平方向と垂直方向の座標です。座標の原点は親ウィンドウに対応しています。

Style はウィンドウのスタイルフラグを保持します。***lpszName*** と ***lpszClass*** はヌルで終わる文字列へのポインタで，それぞれウィンドウの名前とクラス名を指定します。***ExStyle*** はウィンドウの拡張スタイル情報を保持します。

DCB 型 WINDOWS.H

定義
```
typedef struct tagDCB {
    BYTE Id ;
    WORD BaudRate ;
    BYTE ByteSize ;
    BYTE Parity ;
    BYTE StopBits ;
    WORD RlsTimeout ;
    WORD CtsTimeout ;
```

```
    WORD DsrTimeout ;
    BYTE fBinary : 1 ;
    BYTE fRtsDisable : 1 ;
    BYTE fParity : 1 ;
    BYTE fOutxCtsFlow : 1 ;
    BYTE fOutxDsrFlow : 1 ;
    BYTE fDummy : 2 ;
    BYTE fDtrDisable : 1 ;
    BYTE fOutX : 1 ;
    BYTE fInX : 1 ;
    BYTE fPeChar : 1 ;
    BYTE fNull : 1 ;
    BYTE fChEvt : 1 ;
    BYTE fDtrflow : 1 ;
    BYTE fRtsflow : 1 ;
    BYTE fDummy2 : 1 ;
    char XonChar ;
    char XoffChar ;
    WORD XonLim ;
    WORD XoffLim ;
    char PeChar ;
    char EofChar ;
    char EvtChar
    WORD TxDelay ;
} DCB ;
```

シルアル通信デバイスの制御情報を格納する構造体です。***BuildCommDCB, GetCommState, SetCommState*** 関数に使用します。

フィールド ***Id*** メンバは通信デバイスの ID です。上位ビットがセットされているとデバイスはパラレルデバイスです。***LPTx*** マスクと比較してください。上位ビットがセットされなければシリアルポートになります。

BaudRate, ByteSize, Parity, StopBits が定義するのは，ポートの通信パラメータです。***ByteSize*** は各文字のビット数を示し，4 から 8 までの値を取ります。パリティは通信定数であり，***EvenParity, MarkParity, NoParity, OddParity, SpaceParity*** のいずれかです。***StopBits*** も通信定数であり，

OneStopBit, One 5 StopBits, TwoStopBits のいずれかです。通信定数の定義は第7章を参照してください。

RlsTimeout, CtsTimeout, DsrTimeout はそれぞれ RLSD, CTS, DSR 信号に対応します。タイムアウトまでの信号待ち時間をミリ秒単位で示します。

fBinary は，ポートの使用モードがバイナリモードかどうかを示します。バイナリモードでは，データがそのまま変更なしで流れます。非バイナリモードでは，データの終わりを示す ***EofChar*** 文字が使われます。

各ビットは，さまざまなエラー検出用のトグルスイッチになっており，定義は**表 6.2** のとおりです。個々のビットには ***DCB_*** 定数でアクセスします。

表 6.2
DCB ビットフラグ

ビット	**セット時の意味**
fRtsDisable	RTS 無効
fParity	パリティチェック有
fOutxCtsFlow	CTS 送信モニタ
fOutxDsrFlow	DSR 送信モニタ
fDummy	予約済み
fDtrDisable	DTR 無効
fOutX	Xon/Xoff 制御送信
fInX	Xon/Xoff 制御受信
fPeChar	パリティエラー置き換え
fNull	ヌル文字破棄
fChEvt	***EvtChar*** 文字のイベント扱い
fDtrFlow	DTR による受信フロー制御
fRtsFlow	RTS による受信フロー制御
fdummy 2	予約済み

XonChar は送受信の XON 文字の値，***XoffChar*** は XOFF 文字の値を示します。***XonLim*** は受信キューの文字数です。***XoffLim*** は XOFF の転送が行なわれるまでの受信キューの文字数です。

PeChar はパリティエラーの置き換え文字，***EofChar*** はデータの終わりを示す文字，***EvtChar*** はイベントを示す文字を定義します。

TxDelay メンバは現在使用されていません。

DDEACK 型　DDE.H

定義

```
typedef struct {
    unsigned bAppReturnCode : 8,
             reserved : 6,
             fBusy : 1;
    fAck : 1;
} DDEACK;
```

WM_DDE_INITIATE 以外の DDE メッセージに対し ***WM_DDE_ACK*** メッセージのパラメータとして送られる肯定応答を保持します。

フィールド　現在アプリケーションで使える定義ビットは 2 つだけです。***bAppReturnCode*** にアクセスするには ***DDE_APPRETURNCODE*** マスクを使用します。そこにはアプリケーション専用の戻りコードが入ります。***fBusy*** ビットのセットは要求受け入れ不能を，***fAck*** ビットのセットは要求の受け入れを示します。***fAck*** が 0 以外の場合 ***fBusy*** は無意味です。

残りのビットは Windows の予約ビットです。

DDEADVISE 型　DDE.H

定義

```
typedef struct {
    unsigned reserved : 14,
             fDeferUpd : 1,
             fAckReq : 1;
    int cfFormat;
} DDEADVISE;
```

DDE サーバへの要求を保持する構造体で，***WM_DDE_ADVISE*** メッセージの引数で渡されます。

フィールド　***fAckReq*** ビットをセットすると，サーバに対し，***WM_DDE_DATA*** メッセージの ***fAckReq*** ビットをセットして送るよう要求が出ます。これでメッセージフロー制御を実現できます。***fDeferUpd*** ビットを設定すると，サーバに対し，

WM_DDE_DATA メッセージの ***hData*** ハンドルをヌルにして送るよう要求が出ます。これでクライアントは，データの変更を知らされるので，***WM_DDE_REQUEST*** メッセージを送って更新データを要求することができます。

cfFormat メンバには ***CF_*** クリップボード書式定数で，クライアントが望むデータ書式を指定します。

DDEDATA 型　　　　DDE.H

定義

```
typedef struct {
    unsigned unused : 12,
             fResponse : 1,
             fRelease : 1,
             reserved : 1,
             fAckReq : 1;
    int cfFormat;
    BYTE Value [1];
} DDEDATA;
```

アプリケーション間の転送データを保持する構造体です。***WM_DDE_DATA*** メッセージのパラメータとして渡されます。

フィールド

fResponse, fRelease, fAckReq は ***DDE_*** 定数でアクセスします。

fResponse ビットのセットは，クライアントが要求したデータであることを示します。***fRelease*** ビットがセットされると，***WM_DDE_DATA*** メッセージで転送したデータをクライアント側で処理後に解放しなければならないことを示します。1つのビットは予約済みです。***fAckReq*** ビットのセットによって，クライアントアプリケーションからデータ受け取りの肯定応答として ***WM_DDE_ACK*** メッセージを送るようにします。

cfFormat は ***CF_*** クリップボード書式定数を保持するメンバです。クライアントに送られるデータの書式を示します。***Value*** メンバが保持するのは ***cfFormat*** 書式の転送データです。

DDEPOKE 型　　DDE.H

定義
```
typedef struct {
    unsigned unused : 13,
             fRelease : 1,
             fReserved : 2;
    int cfFormat;
    BYTE Value [1];
} DDEPOKE;
```

WM_DDE_POKE メッセージに伴う非要求データを保持する構造体です。

フィールド　***fRelease*** ビットがセットされていれば，受信側が処理後のデータを解放しなければなりません。2 ビット予約されています。***cfFormat*** メンバには ***CF_*** クリップボード書式定数を使い，クライアントの希望するデータ書式を指定します。***Value*** は ***cfFormat*** で指定した書式で転送されるデータを保持します。

DELETEITEMSTRUCT 型　　WINDOWS.H

定義
```
typedef struct tagDELETEITEMSTRUCT {
    WORD CtlType;
    WORD CtlID;
    WORD itemID;
    HWND HWNDItem;
    DWORD itemData;
} DELETEITEMSTRUCT;
```

オーナー描画コンボボックスまたはリストボックスからの削除項目を記述する構造体です。*lParam* パラメータを ***DELETEITEMSTRUCT*** 構造体のポインタとする ***WM_DELETEITEM*** メッセージが項目のオーナーに送られます。

フィールド　***CtlType*** メンバでコントロールの種別を示します。リストボックスの場合は ***ODT_LISTBOX***，コンボボックスの場合は ***ODT_COMBOBOX*** となります。***CtlId*** はボックスのコントロール ID です。***itemID*** は削除項目のインデックス

です。***HWNDItem*** はコントロールのウィンドウハンドルで，***itemData*** は項目の 32 ビット値です。

DEVMODE 型 DRIVEINIT.H

定義

```
typedef struct_devicemode {
    char dmDeviceName [CCHDEVICENAME];
    WORD dmSpecVersion;
    WORD dmDriverVersion;
    WORD dmSize;
    WORD dmDriverExtra;
    DWORD dmFields;
    short dmOrientation;
    short dmPaperSize;
    short dmPaperLength;
    short dmPaperWidth;
    short dmScale;
    short dmCopies;
    short dmDefaultSource;
    short dmPrintQuality;
    short dmColor;
    short dmDuplex;
} DEVMODE;
```

DeviceCapabilities と ***ExtDeviceMode*** によって使用される構造体で，プリンタのデバイスドライバ情報を保持します。

フィールド ***dmDeviceName*** メンバにはヌルで終わる文字列が入り，対応するプリンタ名を示します。***dmSpecVersion*** はデータ仕様のバージョン番号で，現在の番号は 0x0300 となっています。***dmDriverVersion*** はドライバのバージョン番号で開発者が指定します。***dmSize*** は最後の ***dmDriverData*** メンバを含まない構造体サイズです。***dmDriverExtra*** は ***dmDriverData*** メンバのサイズを示します。

dmFields は 32 ビットのビットマップメンバです。***dmFields*** 以降の初期化済みメンバを示します。各ビットが各メンバに対応します。定数を**表 6.3** に一覧表示します。

表 6.3
DEVMODE フィールドフラグ

フラグ	フィールド
DM_COLOR	***dmColor***
DM_COPIES	***dmCopies***
DM_DEFAULTSOURCE	***dmDefaultSource***
DM_DUPLEX	***dmDuplex***
DM_ORIENTATION	***dmOrientation***
DM_PAPERLENGTH	***dmPaperLength***
DM_PAPERSIZE	***dmPaperSize***
DM_PAPERWIDTH	***dmPaperWidth***
DM_PRINTQUALITY	***dmPrintQuality***
DM_SCALE	***dmScale***
DM_SPECVERSION	***dmSpecVersion***

dmOrientation で用紙の向きを指定します。***DMORIENT***_定数でポートレート(縦)かランドスケープ(横)を指定します。***dmPaperSize*** で ***DMPAPER***_定数により用紙サイズを選択します。***dmPaperLength*** と ***dmPaperWidth*** では ***dmPaper*** で指定した用紙の長さと幅をオーバーライドできます。***dmScale*** では ***dmScale***/100 の率で印刷出力の倍率を指定します。値が 75 であれば通常サイズの 75% に縮小されます。***dmCopies*** で印刷枚数を選択します。

dmDefaultSource は用紙を供給するデフォルトのビンを指定します。ビンの指定は ***DMBIN***_定数で行ないます。***dmPrintQuality*** は印刷解像度を指定します。その指定には，デバイス独立の負の ***DMRES***_定数，またはインチ当たりのドット数を表わすデバイス依存の正の数を使います。

dmColor フィールドでカラー印刷またはモノクロ印刷を選択します。***DMCOLOR***_定数でいずれかを指定します。***dmDuplex*** で片面印刷または両面印刷を選択します。***DMDUP***_定数でいずれかを指定します。

DLGITEMTEMPLATE　　WINDOWS.H

DLGITEMTEMPLATE にはヌルで終わる文字列データが埋め込まれており厳密には構造体といえないので，C＋＋に限らず他のどのプログラミング言語でも表現できません。したがって，ダイアログ項目のテンプレートヘッダを構造体ではなく構造体テンプレートとみなします。

DLGITEMTEMPLATE は5つの整数，1つの long int，2つのヌルで終わる文字列，1つのバイト，付加データという順の構成になっています。付加データがない場合もあります。最初の4つの整数 *dtilX, dtilY, dtilCX, dtilCY* は，ダイアログボックスの原点を基点としたコントロールの位置とサイズを指定します。それぞれの値の単位はダイアログテンプレートヘッダ内の該当項目と同じです。

5番目の整数 *dtilID* で，コントロールのダイアログ項目 ID を指定します。ID に続く long int 値 *dtilStyle* はダイアログボックス項目のスタイルを保持します。スタイル情報に続くヌルで終わる文字列は，コントロールの Windows クラス ***BUTTON, EDIT, STATIC, LISTBOX, SCROLLBAR, COMBOBOX*** のいずれかを保持します。クラス名の直後に続くヌルで終わる別の文字列 *dtilText* は，項目のテキストを保持します。

文字列の後には *dtilInfo* と呼ばれるバイトが続き，後続の付加情報のバイト数を示します。値が0であれば付加情報がないので，ダイアログ項目テンプレートを終了します。値が0以外の場合は，次のバイト群である *dtilData* が情報を保持しています。この情報は ***CREATESTRUCT*** 構造体の *lpCreateParams* メンバで ***CreateWindow*** 関数に渡されます。

Windows API 関数 ***CreateDialogIndirect, CreateDialogIndirectParam, DialogBoxIndirect, DialogBoxIndirectParam*** は Windows データ構造体 ***DLGTEMPLATE*** のポインタを引数とします。

DLGTEMPLATE　　WINDOWS.H

DLGTEMPLATE にはヌルで終わる文字列データが埋め込まれており，厳密には構造体でないため，C＋＋に限らず他のどのプログラミング言語でも表現できません。したがって，ダイアログテンプレートは，構造体ではなく構造体テンプレートとみなします。

ダイアログテンプレートは3つの部分で構成されますが，必要なのは最初の ***DLGTEMPLATEHEADER*** だけです。ダイアログテンプレートヘッダの *dtStyle* に ***DS_SETFONT*** 定数があれば，現在のシステムフォント以外のフォントがダイアログボックスで使われます。ダイアログテンプレートヘッダの直後には，フォントを記述する2つの項目，すなわちポイント数を示す符号つきの8ビット値と書体名を示すヌルで終わる文字列を必ず続けます。指定するフォントは，初期化ファイル経由でまたはAPI関数 ***LoadFont*** の呼び出しによってWindowsが使えるものでなければなりません。この結果，***DLGITEMTEMPLATE*** がダイアログボックスの各項目ごとに表示されます。

ダイアログテンプレートを指すポインタはWindows API関数 ***CreateDialogIndirect, CreateDialogIndirectParam, DialogBoxIndirect, DilogBoxIndirectParam*** に引数として渡されます。

DLGTEMPLATEHEADER WINDOWS.H

DLGTEMPLATEHEADER にはヌルで終わる文字列データが埋め込まれており厳密には構造体ではないため，C++に限らず他のどのプログラミング言語でも表現できません。したがって，ダイアログテンプレートヘッダは構造体ではなく構造体テンプレートとみなします。

ダイアログテンプレートヘッダの最初の6つの項目は厳密に定義され，命名されているので以下のように構造体のように見えるかもしれません。

```
Struct DialogTemplateHeader {
    long dtStyle;
    BYTE dtItemCount;
    int dtX;
    int dtY;
    int dtCX;
    int dtCY;
};
```

要するに，ダイアログテンプレート情報のブロックには，まずダイアログボックスのスタイルを定義するlong値を入れてください（***DLGTEMPLATE*** 参照）。この値は ***DS_*** ダイアログスタイル定数の1つ，または複数の組み合わせとしま

す。次のバイト ***dtItemCount*** で，ダイアログボックス内の項目数を指定します。

項目数に続く ***dtX*** と ***dtY*** はそれぞれ，ダイアログボックスの左上隅のX座標とY座標を示します。座標X，Yの原点位置は，ダイアログのスタイルに ***DS_ABSALIGN*** がない場合，親ウィンドウのクライアント領域の原点となります。***dtCX*** はダイアログボックスの幅，***dtCY*** はその高さです。

ダイアログボックスの座標とサイズを指定するときの単位については以下のとおりです。ダイアログ単位は現在のシステムフォントのサイズから得られます。API関数 ***GetDialogBaseUnits*** はピクセル数で基本単位サイズを返します。したがって，ダイアログの座標とサイズは基本単位に対する割合で表わせます。水平方向の構成要素 ***dtX*** と ***dtCX*** は幅の基本単位の4分の1の単位で，垂直方向の構成要素 ***dtY*** と ***dtCY*** は高さの基本単位の8分の1の単位で表わせます。

ダイアログボックスの座標と位置の後に，3つのヌルで終わる文字列 *deMenuName, dtClassName, dtCaptionText* が続きます。それぞれ，ダイアログボックスのメニュー名，Windowsクラス名，タイトルを示す文字列です。

DRAWITEMSTRUCT型 WINDOWS.H

定義
```
typedef struct tagDRAWITEMSTRUCT {
    WORD CtlType;
    WORD CtlID;
    WORD itemID;
    WORD itemAction;
    WORD itemState;
    HWND hwndItem;
    HDC hDC;
    RECT rcItem;
    DWORD itemData;
}   DRAWITEMSTRUCT;
typedef DRAWITEMSTRUCT NEAR *PDRAWITEMSTRUCT;
typedef DRAWITEMSTRUCT FAR *LPDRAWITEMSTRUCT;
```

オーナー描画コントロールの描画データを保持する構造体です。コントロールのオーナーは，この構造体へのポインタを ***WM_DRAWITEM*** メッセージの *lParam* パラメータで受け取ります。

フィールド ***CtlType*** メンバは ***ODT_*** 定数によって指定されるコントロールの種類を示します。***CtlID*** はコントロールの ID 番号です。ただしメニューには使用しません。***itemID*** は項目インデックスのメニュー項目 ID で，コントロールによって決まります。空のリストボックスやコンボボックスは -1 になることがあります。

itemAction メンバは描画の時期と方法を ***ODA_*** 定数で指定します。

itemState メンバは描画後の項目の状態を ***ODS_*** 定数で記述します。***hwndItem*** メンバはコントロールのウィンドウハンドル，または，項目を含むメニューのハンドルです。***hDC*** はコントロール描画に使用されるデバイスコンテキストのハンドルです。

rcItem メンバはデバイスコンテキストのコントロールの境界矩形です。境界矩形の定義は ***RECT*** 構造体で行なわれます。この矩形の境界を越えるオーナー描画コントロール部分はクリップされます。ただし，メニュー項目は矩形境界を越えてもかまいません。

itemData メンバには次の 2 種の値が含まれます。1 つはオーナー描画リストボックス値またはコンボボックス値です。この値を渡すのは次の項目作成メッセージです。

CB_ADDSTRING, CB_INSERTSTRING, LB_ADDSTRING, LB_INSERTSTRING

もう 1 つはメニュー項目の long int 値です。この値を渡すのは項目を挿入した ***InsertMenu*** 呼び出しの *NewItem* パラメータです。***itemData*** はオーナー描画ボタンには定義されていません。

DWORD 型 WINDOWS.H

定義 typedef unsigned long DWORD;

符号なしの 32 ビット整数です。

FARPROC 型 WINDOWS.H

定義 typedef int (FAR PASCAL *FARPROC) () ;

Pascal 呼び出し規約を使う関数のポインタです。
整数を返します。

GLOBALHANDLE 型 WINDOWS.H

定義 typedef HANDLE GLOBALHANDLE ;

HANDLE とまったく同じです。この型をあえて使用するのは，プログラムを理解しやすくするためです。たとえば，ハンドルがグローバルメモリブロックなどのグローバル項目であることを明示します。

HANDLE 型 WINDOWS.H

定義 typedef WORD HANDLE ;

汎用ハンドル型を定義します。

HANDLETABLE 型 WINDOWS.H

定義
```
typedef struct tagHANDLETABLE {
    HANDLE objectHANDLE [1] ;
} HANDLETABLE ;
```

描画ツールなどを多数格納するためのハンドル配列です。

HBITMAP 型　　WINDOWS.H

定義　typedef HANDLE HBITMAP ;

ビットマップのハンドル型です。

HBRUSH 型　　WINDOWS.H

定義　typedef HANDLE HBRUSH ;

ブラシのハンドル型です。

HCURSOR 型　　WINDOWS.H

定義　typedef HANDLE HCURSOR ;

カーソルのハンドル型です。

HDC 型　　WINDOWS.H

定義　typedef HANDLE HDC ;

デバイスコンテキストのハンドル型です。ディスプレイコンテキストもデバイスコンテキストの１つとして，この型の変数に入ります。

HFONT 型　　WINDOWS.H

定義　typedef HANDLE HFONT ;

フォント描画ツールのハンドル型です。

HICON 型　　WINDOWS.H

定義　typedef HANDLE HICON ;

アイコンのハンドル型です。

HMENU 型　　WINDOWS.H

定義　typedef HANDLE HMENU ;

メニューリソースのハンドル型です。

HPALETTE 型　　WINDOWS.H

定義　typedef HANDLE HPALETTE ;

パレットのハンドル型です。

HPEN 型　　WINDOWS.H

定義　typedef HANDLE HPEN ;

ペンのハンドル型です。

HRGN 型　　WINDOWS.H

定義　typedef HANDLE HRGN ;

領域のハンドル型です。

HSTR 型　　WINDOWS.H

定義　typedef HANDLE HSTR ;

文字列のハンドル型です。

HWND 型　　WINDOWS.H

定義　typedef HANDLE HWND ;

ウィンドウのハンドル型です。通常 ObjectWindows のインターフェースオブジェクトが関連する Windows インターフェース要素の管理に用います。API 関数呼び出しの多くはウィンドウハンドルで目的となるウィンドウを示します。

LOCALHANDLE 型　　WINDOWS.H

定義　typedef HANDLE LOCALHANDLE ;

HANDLE とまったく同じです。この型をあえて使用するのは，プログラムを理解しやすくするためです。たとえば，ハンドルがローカルメモリブロックなどのローカル項目であることを明示します。

LOGBRUSH 型　　WINDOWS.H

定義
```
typedef struct tagLOGBRUSH {
    WORD lbStyle ;
    DWORD lbColor ;
    int lbHatch ;
} LOGBRUSH ;
```

CreateBrushIndirect 関数による論理ブラシの作成に関する情報を保持する構造体です。

フィールド　***lbStyle*** メンバが保持するのは ***BS***_ブラシスタイル定数で，ソリッド，消去，ハッチ，パターンのいずれかのブラシを指定します。

lbColor メンバは ***COLORREF*** 構造体です。ブラシスタイルが消去またはパターンであればカラーは無視されます。***BS_DIBPATTERN*** スタイルの場合，下位ワードに ***DIB***_定数のいずれかを入れます。このとき，指定カラーが明示的なものか，または現パレットに対するインデックスかを示します。

lbHatch メンバはハッチスタイルを示します。ブラシのスタイルによって ***lbHatch*** は下記のいずれかになります。

表 6.4 LOGBRUSH ハッチスタイル

スタイル	lbHatch 定数
BS_DIBPATTERN	デバイス独立のパック形式ビットマップのハンドル
BS_HATCHED	ハッチの向きを指定する ***HS***_ハッチスタイル定数
BS_HOLLOW	無視
BS_PATTERN	パターンを定義するビットマップのハンドル
BS_SOLID	無視

定義

```
typedef struct tagLOGFONT {
    int lfHeight;
    int lfWidth;
    int lfEscapement;
    int lfOrientation;
    int lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE ifClipPrecision;
    BYTE ifQuality;
    BYTE lfPitchAndFamily;
    BYTE lfFaceName [LF_FACESIZE] ;
} LOGFONT;
```

CreateFontIndirect 関数に使用する論理フォントの属性を保持する構造体です。

フィールド

lfHeight と ***lfWidth*** はフォントの平均の高さと幅を示します。***lfEscapement*** と ***lfOrientation*** はテキストの出力方向と傾斜角度です。両方とも 1/10 度単位の角度で示され，X 軸から左回りに表現されます。

lfWeight はインクピクセル数 (inked pixels) によるフォントの太さで 1000 までの値で示します。一般的に 400 は通常のタイプフェース，700 はボールド体です。実際の値はフォントの種類によって異なります。値を 0 とすればデフォルトの太さになります。

lfItalic, lfUnderline, lfStrikeOut メンバは通常 0 です。0 以外にすると，それぞれイタリックフォント，下線つきフォント，ストライクアウトつきフォントになります。***lfCharSet*** は ***ANSI_CHARSET, OEM_CHARSET, SYMBOL_CHARSET, SHIFTJIS_CHARSET, HANGEUL_CHARSET*** のいずれかの定義済みの文字セットとします。他の文字セットの定義も可能です。

lfOutPrecision メンバには ***OUT_*** のフォント精度フラグを含みます。デフォ

ルトは ***OUT_DEFAULT_PRECIS*** です。***lfClipPrecision*** は ***CLIP_*** のフォント精度フラグに定義されるフォントのクリッピング精度を指定します。デフォルト値は ***CLIP_DEFAULT_PRECIS*** です。

lfQuality メンバに含まれるのはフォント品質フラグです。***DEFAULT_QUALITY, DRAFT_QUALITY, PROOF_QUALITY*** の3種類があります。***lfPitchAndFamily*** は次の2種類のフラグの組み合わせです。1つは ***DEFAULT_PITCH, FIXED_PITCH, VARIABLE_PITCH*** のいずれかのフォントピッチフラグです。もう1つは ***FF_ROMAN, FF_SCRIPT*** などのフォントファミリーフラグです。***lfFaceName*** はヌルで終わる文字列のフォント名を保持します。***lfFaceName*** を NULL にすると，デフォルトの書体が使用されます。

LOGPALETTE 型 WINDOWS.H

L
Typ

定義

```
typedef struct tagLOGPALETTE {
WORD palVersion;
WORD palNumEntries;
PALETTEENTRY palPalEntry [1];
} LOGPALETTE;
```

CreatePalette 関数などに使用される論理パレットの定義データを保持する構造体です。

フィールド

palVersion メンバは構造体の Windows バージョンを示します。現バージョンは 0x0300 となっています。***palNumEntries*** はパレットのエントリ数です。***palPalEntry*** は ***PALETTEENTRY*** 構造体の配列で，配列の各要素がパレットの各エントリに対応します。

LOGPEN 型　WINDOWS.H

定義
```
typedef struct tagLOGPEN {
    WORD lopnStyle ;
    POINT lopnWidth ;
    DWORD lopnColor ;
} LOGPEN ;
```

論理ペンの属性を格納する構造体で，***CreatePenIndirect*** 関数に使用されます。

フィールド　***lopnStyle*** メンバは ***PS*** _定数のペンスタイルを保持します。***lopnWidth*** は論理単位で表わしたペン幅です。***lopnColor*** はペンカラーです。

MDICREATESTRUCT 型　WINDOWS.H

定義
```
typedef struct tagMDICREATESTRUCT {
    LPSTR szClass ;
    LPSTR szTitle ;
    HANDLE hOwner ;
    int x, y ;
    int cx, cy ;
    LONG style ;
    LONG lParam ;
} MDICREATESTRUCT ;
```

MDI 子ウィンドウの作成データを保持する構造体です。***WM_CREATE*** メッセージの *lParam* パラメータは ***CREATESTRUCT*** 構造体のポインタを保持します。この ***CREATESTRUCT*** 構造体の ***lpCreateParams*** メンバが ***WM_MDICREATE*** メッセージによって指定された ***MDICREATESTRUCT*** 構造体のポインタになっています。

フィールド　***szClass*** メンバは子ウィンドウクラスのポインタです。***szTitle*** はウィンドウタイトルのポインタです。***hOwner*** はウィンドウ作成アプリケーションのイン

スタンスハンドルです。

x と ***y*** は子ウィンドウの初期X座標とy座標です。***cx*** と ***cy*** は初期の幅と高さです。***CW_USEDEFAULT*** とするとデフォルトの座標，幅，高さになります。

style メンバが保持するのはウィンドウの追加スタイルで，***WS_MINIMIZE, WS_MAXIMIZE, WS_HSCROLL, WS_VSCROLL*** のいずれかです。

lParam はアプリケーションによります。

MEASUREITEMSTRUCT 型　　WINDOWS.H

定義

```
typedef struct tagMEASUREITEMSTRUCT {
    WORD CtlType;
    WORD CtlID;
    WORD itemID;
    WORD itemWidth;
    WORD itemHeight;
    DWORD itemData;
} MEASUREITEMSTRUCT;
```

オーナー描画コントロールの寸法を保持する構造体です。この構造体は ***WM_MEASUREITEM*** メッセージの *lParam* パラメータにポインタとして入ります。***MEASUREITEM*** メンバを適切に埋めないと，オーナー描画コントロールの動作に支障が起きます。

フィールド

CtlType は ***ODT*** _定数で指定するコントロール種別です。***CtlID*** はコントロールのID番号です。これはメニューには使用しません。***itemID*** はコントロールごとの項目インデックスのメニュー項目IDになります。

itemWidth と ***itemHeight*** メンバは項目の幅と高さです。

itemData メンバには次の2種類の値のいずれかが含まれます。1つはオーナー描画リストボックス値またはコンボボックス値です。この値を渡すのは次の項目作成メッセージです。

CB_ADDSTRING, CB_INSERTSTRING, LB_ADDSTRING, LB_INSERTSTRING

もう1つはメニュー項目の long int 値です。この値を渡すのは項目を挿入した ***InsertMenu*** 呼び出しの *NewItem* パラメータです。***itemData*** はオーナー描画ボタンには定義されていません。

MENUITEMTEMPLATE 型　WINDOWS.H

定義

```
typedef struct {
    WORD mtOption ;
    WORD mtID ;
    LPSTR mtString ;
} MENUITEMTEMPLATE ;
```

メニュー項目の情報を保持する構造体です。これをいくつか連結にするとメニュー項目の一覧リストになります。このリストを ***MENUITEMTEMPLATE-HEADER*** と組み合わせると完全なメニューテンプレートになります。

フィールド

mtOption メンバはメニュー項目の型と状態を定義します。***MF_GRAYED*** や ***MF_POPUP*** などの ***MF_*** メニューフラグ定数の1つ，または組み合わせを含みます。

mtID メンバはメニュー項目の ID コードを保持します。別のメニューを呼び出すポップアップメニュー項目には，***mtID*** メンバがありません。***mtString*** メンバはメニュー項目名を指定するヌルで終わる文字列を保持します。

MENUITEMTEMPLATEHEADER 型　WINDOWS.H

定義

```
typedef struct {
    WORD versionNumber ;
    WORD offset ;
} MENUITEMTEMPLATEHEADER ;
```

メニューテンプレートのヘッダとして用いる構造体で，そのテンプレートには ***MENUITEMTEMPLATE*** 構造体のリストのオフセットを含みます。

フィールド　テンプレートヘッダの ***versionNumber*** メンバは，メニューテンプレートのバージョン番号である 0 を保持します。***offset*** メンバは，ヘッダからメニュー項目リストの先頭までのオフセットをバイト数で渡します。

METAFILEPICT 型　　WINDOWS.H

定義

```
typedef struct tagMETAFILEPICT {
    int mm ;
    int xExt ;
    int yExt ;
    HANDLE hMF ;
} METAFILEPICT ;
```

クリップボードを通したメタファイルデータ交換に使用されるメタファイルピクチャの書式を定義します。

フィールド　***mm*** メンバはピクチャ描画のマッピングモードを保持します。***xExt*** と ***yExt*** はピクチャ描画用矩形領域の幅と高さです。ただしマッピングモードが ***MM_ISOTROPIC*** または ***MM_ANISOTROPIC*** の場合，メンバには推奨サイズが示されます。***hMF*** はメモリメタファイルのハンドルです。

METAHEADER 型　　WINDOWS.H

定義

```
typedef struct tagMETAHEADER {
    WORD mtType ;
    WORD mtHeaderSize ;
    WORD mtVersion ;
    DWORD mtSize ;
    WORD mtNoObjects ;
    DWORD mtMaxRecord ;
    WORD mtNoParamaters ;
} METAHEADER ;
```

メタファイルのヘッダ書式を定義する構造体です。メタファイルを構成するのは，この構造体とそれに続く ***MEASURERECORD*** 型などのメタファイル構造体の並びです。

フィールド　***mtType*** メンバの値には2種類あります。メタファイルの所在がメモリであれば1，ディスク上であれば2です。

mtHeaderSize メンバはヘッダのサイズをバイト数で示します。***mtVersion*** は Windows のバージョン番号で現バージョンの3.0は0x0300になっています。***mtSize*** はワード数によるファイルのサイズです。

mtNoObjects メンバはメタファイルに格納可能な最大オブジェクト数，***mtMaxRecord*** はメタファイルの最大構造体のサイズをワード数で示します。

mtNoParameters は使用しません。

METARECORD 型　　WINDOWS.H

定義
```
typedef struct tagMETARECORD {
    DWORD rdSize;
    WORD rdFunction;
    WORD rdParm [1];
} METARECORD;
```

代表的なメタファイル構造体を定義する構造体です。メタファイル構造体のリストはメタファイルのヘッダの後に続きます。

フィールド　***rdSize*** メンバは構造体のサイズをワード単位で指定します。***rdFunction*** は関数番号を ***META_*** 定数で指定します。***rdParm*** は関数への ***WORD*** 型引数の配列で，関数に渡される順と逆順で格納されます。

MSG 型　　WINDOWS.H

定義
```
typedef struct tagMSG {
    HWND hwnd;
```

```
    WORD message ;
    WORD wParam ;
    DWORD lParam ;
    DWORD time ;
    POINT pt ;
} MSG ;
```

アプリケーションに転送されるメッセージデータを保持する構造体です。情報は該当する転送先のウィンドウで処理されます。この構造体のメンバに含まれるのはアプリケーション関連の重要情報です。Windows のメッセージとパラメータの一覧表示は第 5 章『**Windows メッセージ**』にあります。

フィールド　***hwnd*** メンバはメッセージの送り先ウィンドウのハンドルです。***message*** はメッセージ番号です。

wParam と ***lParam*** メンバはウィンドウに対するワード長と long int 長の情報です。この情報内容は ***message*** の値によって異なります。各メッセージのパラメータ値の説明は第 5 章にあります。

time メンバはメッセージ送出時間，***pt*** はメッセージ送出時の画面上のカーソル位置を保持します。

MULTIKEYHELP 型　　WINDOWS.H

定義

```
typedef struct tagMULTIKEYHELP {
    WORD mkSize ;
    BYTE mkKeyList ;
    BYTE szKeyPhrase [1] ;
} MULTIKEYHELP ;
```

Windows のヘルプシステムに使用されるキーワードテーブルへのインデックスと検索対象の句を保持するレコードです。

フィールド　***mkSize*** メンバはレコードの長さを保持します。***mkKeyList*** は検索対象のキーワードテーブルを指定する文字です。***szKeyPhrase*** はヌルで終わる文字列で，検索対象のキーワードを保持します。

OFSTRUCT 型　　WINDOWS.H

定義

```
typedef struct tagOFSTRUCT {
    BYTE cBytes ;
    BYTE fFixedDisk ;
    WORD nErrCode ;
    BYTE reserved [4] ;
    BYTE szPathName [128] ;
} OFSTRUCT ;
```

ファイルのオープン時に読み込まれたファイル情報を保持する構造体です。

フィールド　***cBytes*** メンバは構造体長を保持します。***fFixedDisk*** はファイルがハードディスク上にあれば 0 以外，その他の場合は 0 です。***nErrCode*** は ***OpenFile*** 関数の失敗によって -1 が返る場合に DOS エラーコードを保持します。***reserved*** の 4 バイトは Windows が将来使うために予約されています。

szPathName メンバはヌルで終わる文字列で，ファイルのフルパス名を保持します。

PAINTSTRUCT 型　　WINDOWS.H

定義

```
typedef struct tagPAINTSTRUCT {
    HDC hdc ;
    BOOL fErase ;
    RECT rcPaint ;
    BOOL fRestore ;
    BOOL fIncUpdate ;
    BYTE rgbReserved [16] ;
} PAINTSTRUCT ;
```

ウィンドウのクライアント領域の描画情報を保持する構造体です。大部分は Windows による内部使用のための予約メンバですが，ユーザが使用できるメンバもあります。

フィールド　*hdc* メンバは描画が行なわれるディスプレイコンテキストのハンドルです。*fErase* はバックグラウンドの再描画の有無を示します。0であれば再描画されていません。*rcPaint* は描画が行なわれる矩形領域を定義します。

他は Windows 用の予約メンバです。

PALETTEENTRY 型　WINDOWS.H

定義
```
typedef struct tagPALATTEENTRY {
    BYTE peRed ;
    BYTE peGreen ;
    BYTE peBlue ;
    BYTE peFlags ;
} PALETTEENTRY ;
```

論理パレットのエントリを定義する構造体型です。*LOGPALETTE* によるエントリ定義はその1例です。

フィールド　*peRed*, *peGreen*, *peBlue* の各メンバは，それぞれパレットエントリの赤，緑，青の輝度を表わします。*peFlags* にはパレットの使用法に関する情報が入ります。このメンバが取り得る値は0または *PC_EXPLICIT*, *PC_NOCOLLAPSE*, *PC_RESERVED* のいずれかの *PC_* フラグです。

P
Typ

PATTERN 型　WINDOWS.H

定義
```
typedef LOGBRUSH PATTERN ;
```

LOGBRUSH の別名です。論理ブラシによるパターン塗りつぶしを区別するために *PATTERN* の名称を使います。

PINT 型　WINDOWS.H

定義
```
typedef int PINT ;
```

符号つきの 16 ビット整数値を指すポインタを定義します。

POINT 型　WINDOWS.H

定義
```
typedef struct tagPOINT  {
      int x ;
      int y ;
}  POINT ;
```

画面またはウィンドウ上のポイントの X 座標と y 座標を指定します。

RECT 型　WINDOWS.H

定義
```
typedef struct tagRECT  {
      int left ;
      int top ;
      int right ;
      int bottom ;
}  RECT ;
typedef RECT *PRECT ;
typedef RECT NEAR *NPRECT ;
typedef RECT FAR *LPRECT ;
```

左上隅と右下隅を指定した矩形領域を定義します。

フィールド　***left*** と ***top*** メンバは矩形の左上隅の X 座標と Y 座標です。***right*** と ***bottom*** は右下隅の X 座標と Y 座標です。

➡ 矩形領域の幅も高さも 32,768 を超えないものとします。

RGBQUAD 型 WINDOWS.H

定義
```
typedef struct tagRGBQUAD {
    BYTE rgbBlue;
    BYTE rgbGreen;
    BYTE rgbRed;
    BYTE rgbReserved;
} RGBQUAD;
```

ビットマップに使用される RGB カラーデータを保持する構造体です。***BITMAPINFO*** 構造体の ***bmiColors*** メンバなどに使用されます。

フィールド ***rgbBlue, rgbGreen, rgbRed*** はそれぞれビットマップのピクセルエントリの青，緑，赤の輝度を表わします。***rgbReserved*** は不要なので 0 にセットします。

RGBTRIPLE 型 WINDOWS.H

定義
```
typedef struct tagRGBTRIPLE {
    BYTE rgbtBlue;
    BYTE rgbtGreen;
    BYTE rgbtRed;
} RGBTRIPLE;
```

ビットマップの RGB カラー輝度を指定する構造体です。***BITMAPCOREINFO*** 構造体の ***bmciColors*** メンバなどに使用されます。

フィールド ***rgbBlue, rgbGreen, rgbRed*** はそれぞれビットマップのピクセルエントリの青，緑，赤の輝度を表わします。

R Type

定義
```
typedef struct tagTEXTMETRIC {
    int tmHeight;
    int tmAscent;
    int tmDescent;
    int tmInternalLeading;
    int tmExternalLeading;
    int tmAveCharWidth;
    int tmMaxCharWidth;
    int tmWeight;
    int tmItalic;
    BYTE tmUnderlined;
    BYTE tmStruckOut;
    BYTE tmFirstChar;
    BYTE tmLastChar;
    BYTE tmDefaultChar;
    BYTE tmBreakChar;
    BYTE tmPitchAndFamily;
    BYTE tmCharSet;
    int tmOverhang;
    int tmDigitizedAspectX;
    int tmDigitizedAspectY;
} TEXTMETRIC;
```

この構造体のメンバには，ディスプレイコンテキストのマッピングモードによる物理フォントが記述されます。***GetDeviceCaps*** 関数と ***GetTextMetrics*** 関数に使用します。

フィールド　***tmHeight*** メンバはフォントの文字の高さです。ベースラインより上の ***tmAscent*** が示す高さとベースラインより下の ***tmDescent*** が示す高さの合計になっています。

tmInternalLeading と ***tmExternalLeading*** メンバは ***tmHeight*** を越えたスペースを示します。***tmInternalLeading*** は境界の内側のスペース，***tmExternalLeading*** はテキストの行間のスペースです。両メンバとも0にセットできます。

tmAveCharWidth と ***tmMaxCharWidth*** はフォントの文字の平均幅と最大幅です。

tmWeight はフォントの太さを示します。

tmItalic, ***tmUnderlined,*** ***tmStruckOut*** は0以外であれば，イタリック，下線つき，ストライクアウトつきのテキストをそれぞれ示します。

フォントの定義文字の範囲は ***tmFirstChar*** と ***tmLastChar*** で示します。範囲外の文字は ***tmDefaultChar*** の指定文字に置き換えられます。***tmBreakChar*** はワードブレークを表わす文字で，位置合わせに使われます。

フォントのピッチ，ファミリー，文字セットの定義は ***tmPitchAndFamily*** と ***tmCharSet*** によって行なわれます。***tmPitchAndFamily*** の最下位ビットでフォントのピッチを指定します。ビットが0であれば固定ピッチを，1であれば可変ピッチです。上位4ビットでフォントファミリーを指定します。このファミリーの設定または取得は ***FF_***フォントファミリーフラグで行ないます。文字セットの設定は ***tmCharSet*** のフォント文字セットフラグによって行ないます。

tmOverhang メンバに保持されるのは文字列当たりの追加幅です。ノーマルフォントからボールドフォントを合成する場合などに生ずる文字列幅の増加を処理します。

フォントを使用する対象デバイスの水平および垂直アスペクト比は ***tmDigitizedAspectX*** と ***tmDigitizedAspectY*** に示されます。

WNDCLASS 型

定義

```
typedef struct tagWNDCLASS {
    WORD style;
    LONG (FAR PASCAL *lpfnWndProc) (HWND, WORD, LONG, LONG);
    int cbClsExtra;
    int cbWndExtra;
    HANDLE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPSTR lpszMenuName;
    LPSTR lpszClassName;
} WNDCLASS;
```

ウィンドウクラスの属性を保持する構造体です。この属性は登録属性とも呼ばれ，***RegisterClass*** 関数で登録されます。

フィールド

style はクラススタイルを保持します。***CS***_クラススタイル定数の組み合せも保持できます。

lpfnWndProc メンバはメッセージの受け取りと処理を行なうウィンドウ関数へのポインタです。

cbClsExtra は ***WNDCLASS*** 構造体の末尾に割り当てられる追加バイト数です。このバイトはクラスの追加バイトと呼ばれます。アクセスするには ***GetWindowLong*** または ***GetWindowWord*** 関数，設定を行なうには ***SetWindowLong*** または ***SetWindowWord*** 関数を使用します。

cbWndExtra は各ウィンドウインスタンスの後に割り当てられる追加バイト数です。

hInstance はクラスモジュールを示すインスタンスハンドルです。0 以外の値に限ります。

hIcon はクラスアイコンのハンドル，***hCursor*** はクラスカーソルのハンドル，***hbrBackground*** はバックグラウンド色のハンドルです。バックグラウンド色のカラー値は，***COLOR***_定数の標準システムカラーの値に 1 を加えたものとします。またはバックグラウンド描画用ブラシのハンドルをバックグラウンド色とします。***hbrBackground*** を 0 とすると，クライアント領域の描画とともに必

ずバックグラウンドも描画します。バックグラウンドの再描画が必要かどうかは，***WM_ERASEBKGND*** メッセージを処理します。または，***BeginPaint*** が作成した構造体である ***PAINTSTRUCT*** の ***fErase*** メンバを確認します。

lpszMenuName はクラスメニューのリソース名，***lpszClassName*** はクラス名で，ともにヌルで終わる文字列へのポインタです。

WORD 型　　WINDOWS.H

定義　typedef unsigned int WORD ;

符号なしの 16 ビット整数のポインタを定義します。

W
Typ

第7章

Windows定数

この章では，Windows関数を呼び出すときとWindowsメッセージを受け取るときによく使われる定数を定義します。関数とメッセージの詳細については，第4章と第5章を参照してください。定数は用途が共通するものを1グループにまとめ，このグループのアルファベット順に記載します。

識別子のプレフィクスごとにまとめられるものは，そのプレフィクスのアルファベット順に記載します。たとえば，***COLOR_***というプレフィクスで始まる定数の場合は，「**COLOR_システムカラーコード**」というグループにまとめます。共通のプレフィクスがない定数は，機能別にまとめます。たとえば，***COMPLEXREGION, NULLREGION, SIMPLEREGION***は，すべてグラフィック領域の種類を指定するものであり，同じ関数の引数として使われるものなので「**Region(領域) フラグ**」という項目の下にまとめます。

Background Mode（バックグラウンドモード）

ハッチブラシでテキストまたはグラフィックスを描くときの背景を指定する定数です。***GetBkMode***関数と***SetBkMode***関数で使用されます。

表7.1
バックグラウンドモード

定数	意味
OPAQUE	現在のバックグラウンド色で背景を塗りつぶします。
TRANSPARENT	背景を塗りつぶしません。

BI_ビットマップ圧縮定数

CreateDIBitmap の呼び出しで渡されるWindowsレコード型 *TBITMAPINFOHEADER* の *biCompression* フィールドの値として使います。

表 7.2
ビットマップ圧縮定数

定数	意味
BI_RGB	ビットマップは圧縮されません。
BI_RLE8	ビットマップは1ピクセル当り8ビットのランレングスでコード化された形式を使います。
BI_RLE4	ビットマップは1ピクセル当り4ビットのランレングスでコード化された形式を使います。

BN_ボタン通知コード

ボタンコントロールからの *WM_COMMAND* メッセージで渡される制御通知コードです。ボタン操作を知らせます。

表 7.3
ボタン通知コード

定数	意味
BN_CLICKED	ボタンのクリックを親ウィンドウに知らせます。
BN_DOUBLECLICKED	ボタンのダブルクリックを親ウィンドウに知らせます。この定数が使われるのは *BS_OWNERDRAW* スタイルのボタンとラジオボタンに限ります。

BS_ブラシスタイル

論理ブラシスタイルを指定する定数です。*CreateBrushIndirect* 関数で渡される *LOGBRUSH* レコードの *lbStyle* フィールドに使用します。

表 7.4
ブラシスタイル

定数	意味
BS_DIBPATTERN	デバイス独立のビットマップが指定するブラシパターン。
BS_HATCHED	ハッチブラシ。
BS_HOLLOW	空白のブラシ。
BS_NULL	同上。
BS_PATTERN	メモリビットマップが指定するブラシパターン。
BS_SOLID	ソリッドブラシ。

BS_ボタンスタイル

CreateWindow と ***CreateWindowEx*** の両関数が作成するボタンのスタイルを指定します。

表 7.5
ボタンスタイル

定数	意味
BS_3STATE	チェック，チェック解除，淡色表示の 3 つの状態を選択できるボックスです。通常，淡色表示はボックスの使用不可状態を表わします。
BS_AUTO3STATE	***BS_3STATE*** と基本的に同じです。ただし，クリックすると 3 つの状態が自動的に切り換わります。
BS_AUTOCHECKBOX	***BS_CHECKBOX*** と基本的に同じです。ただし，クリックすると状態が自動的に切り換わります。
BS_AUTORADIOBUTTON	***BS_RADIOBUTTON*** と基本的に同じです。ただし，クリックされたボタン以外のチェックマークはすべて除去されます。
BS_CHECKBOX	チェックとチェック解除の選択ができるボックスです。ボックスの右側に付属テキストが表示されます。
BS_DEFPUSHBUTTON	***BS_PUSHBUTTON*** と基本的に同じです。ただし，このボタンはデフォルトボタンなので，キーボードかマウスで別のボタンやボックスを選択しなければ，自動的に選択されます。
BS_GROUPBOX	他ボタンをひとまとめにするボックスです。ボックスの左上隅に表示されます。
BS_LEFTTEXT	***BS_3STATE***，***BS_CHECKBOX***，***BS_RADIOBUTTON*** のいずれかと併用すると，付属テキストの表示位置がボックスまたはボタンの位置が右側ではなく左側になります。

表 7.5
ボタンスタイル（つづき）

定数	意味
BS_OWNERDRAW	オーナーが描画するボタンです。親ウィンドウは，*WM_COMMAND* メッセージで送られる通常の通知コードの他に，ボタンの描画，反転，無効化の要求も受け取ります。
BS_PUSHBUTTON	テキストが内側に表示されるボタンです。
BS_RADIOBUTTON	チェックまたはチェック解除を行なう小さな円形のボタンです。付属テキストはボタンの右側に表示されます。ラジオボタンは通常グループになっており，その内 1 つだけをチェックできます。

CB_コンボボックス戻り値

CB_ADDSTRING などの Windows コンボボックスメッセージから返される値です。負の値はエラーを示します。

表 7.6
コンボボックス戻り値

定数	意味
CB_ERR	エラー発生のため処理は実行されなかった。
CB_ERRSPACE	コンボボックスに充分なスペースがないので，処理を実行できない。
CB_OKAY	エラーなし。

CBM_CreateDIBitmap 初期化定数

CreateDIBitmap 関数の *Usage* 引数で渡された *CBM_INIT* は，メモリビットマップを初期化することを示します。

CBN_コンボボックス通知コード

コンボボックスコントロールからの ***WM_COMMAND*** メッセージで渡された制御通知コードです。コンボボックスで行なわれた操作を知らせます。

表 7.7
コンボボックス通知コード

定数	意味
CBN_DBCLK	コンボボックスのリストボックス項目がマウスでダブルクリックされたことを，親ウィンドウに知らせます。
CBN_DROPDOWN	コンボボックスのリストボックスが開くことを，親ウィンドウに知らせます。
CBN_EDITCHANGE	エディットコントロールのテキストが変更されたことを，コンボボックスの親ウィンドウに知らせます。
CBN_EDITUPDATE	エディットコントロールの変更後のテキストを再表示することを，コンボボックスの親ウィンドウに知らせます。
CBN_ERRSPACE	システムがメモリ不足になったことを，コンボボックスの親ウィンドウに知らせます。
CBN_KILLFOCUS	コンボボックスから入力フォーカスが無くなったことを，コンボボックスの親ウィンドウに知らせます。
CBN_SELCHANGE	リストボックス項目の選択が変更されたことを，コンボボックスの親ウィンドウに知らせます。
CBN_SETFOCUS	コンボボックスに入力フォーカスを得たことを，コンボボックスの親ウィンドウに知らせます。

CBS_コンボボックススタイル

CreateWindow と ***CreateWindowEx*** の両関数で作成するコンボボックスのスタイルを指定する定数です。

表 7.8
コンボボックススタイル

定数	意味
CBS_AUTOHSCROLL	ユーザーが行末に文字をタイプすると，エディットコントロール内のテキストが右にスクロールされます。このスタイルの設定がないと，エディットコントロール境界外にテキストを入力できません。
CBS_DROPDOWN	***CBS_SIMPLE*** と基本的に同じです。ただし，選択フィールドの隣りのアイコンが選択されないと，リストボックスは表示されません。
CBS_DROPDOWNLIST	***CBS_DROPDOWN*** と基本的に同じです。ただし，エディットコントロールではなくスタティックなテキスト項目によって，現在の選択項目が表示されます。
CBS_HASSTRINGS	***CBS_OWNERDRAWFIXED*** または ***CBS_OWNERDRAW VARIABLE*** と併用できます。このスタイルのコンボボックスでは，エントリに文字列を使用します。文字列はシステムによって管理されるため，***CB_GETLBTEXT*** によって検索することができます。
CBS_NOINTEGRALHEIGHT	このスタイルのコンボボックスのサイズは，作成時に指定されます。通常コンボボックスのサイズは変更可能であり，項目を一部隠すこともできます。
CBS_OEMCONVERT	***CBS_SIMPLE*** または ***CBS_DROPDOWN*** スタイルと併用できます。エディットコントロールに入力された各文字を，ANSI 文字から OEM 文字に変換し，再び ANSI に戻します。これで，リストボックスの項目やエディットコントロールのテキストに ***AnsiToOem*** 関数を使用できます。このスタイルは，ファイル名を扱うコンボボックスに便利です。
CBS_OWNERDRAWFIXED	このスタイルのコンボボックスは，オーナーが描画しなければなりません。リストボックスの項目はすべて同一の高さです。

表 7.8
コンボボックススタイル（つづき）

定数	意味
CBS_OWNERDRAWVARIABLE	このスタイルのコンボボックスは，オーナーが描画しなければなりません。リストボックスの項目は高さが異なります。
CBS_SIMPLE	このスタイルでは，リストボックスが常に表示されます。リストボックスの現選択項目は，エディットコントロールに表示されます。
CBS_SORT	このスタイルでは，リストボックスがソートされます。ソート順は ***CBS_OWNERDRAWFIXED*** や ***CBS_OWNERDRAWVARIABLE*** スタイルのコンボボックスと異なってもかまいません。

CC_曲線作成機能

デバイスの曲線作成機能を表わす定数です。

表 7.9
曲線作成機能定数

定数	意味
CC_NONE	曲線のサポートなし。
CC_CHORD	弦。
CC_CIRCLES	円。
CC_ELLIPSES	楕円。
CC_INTERIORS	インテリア。
CC_PIE	パイ型。
CC_STYLED	スタイル曲線。
CC_WIDE	太い曲線。
CC_WIDESTYLED	太いスタイル曲線。

CCHDEVICENAME デバイス名長定数

この定数によって，デバイス名を 32 文字以下に限定します。

CE_通信エラーフラグ

GetCommError 関数から返される通信エラーを指定するフラグです。

表 7.10
通信エラーフラグ

定数	意味
CE_BREAK	ブレーク検出。
CE_STSTO	送信可タイムアウト。
CE_DNS	LPTx デバイスが未選択。
CE_DSRTO	データセットレディ時間切れ。
CE_FRAME	フレーム指示エラー。
CE_IOE	LPTx 入出力エラー。
CE_MODE	要求モードのサポートなし，または通信識別子無効。
CE_OOP	LPTx 用紙切れ。
CE_OVERRUN	受信文字の重複によって読解不能。
CE_PTO	LPTx タイムアウト。
CE_RLSDTO	受信ライン信号検出タイムアウト。
CE_RXOVER	受信待ち行列オーバーフロー。
CE_RXPARITY	パリティエラー。
CE_TXFULL	送信待ち行列がいっぱい。

CF_クリップボード書式

Windows クリップボードにデータを受け渡しするときのクリップボード書式を識別する定数です。ダイナミックデータエクスチェンジ（DDE）プロトコルにも使用されます。***EnumClipboardFormats, GetClipboardData, GetClipboardFormatName, GetPriorityClipboardFormat, IsClipboardFormatAvailable, SetClipboardData*** の各関数で使います。

表 7.11
クリップボード書式

定数	意味
CF_BITMAP	Windows ビットマップ。
CF_DIB	Windows デバイス独立ビットマップ。
CF_DIF	Software Arts 社の Data Interchange Format。
CF_DSPBITMAP	プライベートビットマップ書式の表示情報。
CF_DSPMETAFILEPICT	プライベートメタファイル書式の表示情報。
CF_DSPTEXT	プライベートテキスト書式の表示情報。
CF_METAFILEPICT	Windows メタファイル。
CF_OEMTEXT	OEM 文字セット使用のテキスト。
CF_OWNERDISPLAY	オーナー表示書式。
CF_PALLETE	特定色を使用するクリップボードデータ用のカラーパレット。
CF_PRIVATEFIRST	プライベートデータ書式のデータ列の最初。この書式のデータへのハンドルは自動的に解放されないのでクローズ前にアプリケーション側で解放してください。
CF_PRIVATELAST	プライベートデータ書式のデータ列の最後。
CF_SYLK	Microsoft SYLK 形式。
CF_TEXT	テキスト。
CF_TIFF	Tag Image File Format。

CLIP_フォントクリッピング精度フラグ

CreateFont 関数で作成されるフォントのクリッピング精度を指定する定数です。クリッピング領域内に収まらないテキストの表示形態を指定します。

CLIP_CHARACTER_PRECIS
CLIP_DEFAULT_PRECIS
CLIP_STROKE_PRECIS

COLOR_システムカラーコード

Windows ユーザーインターフェースのアスペクトを指定して，*SetSysColors* 関数でプログラマ自身が色をセットするのに使われます。

表 7.12
システムカラーコード

定数	意味
COLOR_ACTIVEBORDER	アクティブウィンドウ枠。
COLOR_ACTIVECAPTION	Borland C++ウィンドウキャプション。
COLOR_APPWORKSPACE	MDI フレームウィンドウのバックグラウンド。
COLOR_BACKGROUND	Windows デスクトップ。
COLOR_BTNFACE	プッシュボタンの表面。
COLOR_BTNSHADOW	プッシュボタンの輪郭。
COLOR_BTNTEXT	プッシュボタンのテキスト。
COLOR_CAPTIONTEXT	キャプションテキスト，サイズボックス，スクロールバー矢印ボックス。
COLOR_GRAYTEXT	淡色表示のテキスト。
COLOR_HIGHLIGHT	コントロールの選択項目。
COLOR_HIGHLIGHTTEXT	コントロールの選択項目のテキスト。
COLOR_INACTIVEBORDER	アクティブでないウィンドウの枠。
COLOR_INACTIVECAPTION	アクティブでないウィンドウのキャプション。
COLOR_MENU	メニューバーのバックグラウンド。
COLOR_MENUTEXT	メニューバーのテキスト。
COLOR_SCROLLBAR	スクロールバーのバックグラウンド。

表 7.12
システムカラーコード(つづき)

定数	意味
COLOR_WINDOW	ウィンドウのバックグラウンド。
COLOR_WINDOWFRAME	ウィンドウフレーム。
COLOR_WINDOWTEXT	ウィンドウ内のテキスト。

Comm(通信)コンフィギュレーション定数

以下の定数は,***DCB*** 構造体の ***Parity*** フィールドで使います。シリアル通信デバイスに使われるパリティのタイプを指定します。

表 7.13
通信パリティ定数

定数	パリティ設定
EVENPARITY	偶数。
MARKPARITY	マーク。
NOPARITY	パリティなし。
ODDPARITY	奇数。
SPACEPARITY	スペース。

以下の定数は,***DCB*** 構造体の ***StopBits*** フィールドで使います。シリアル通信デバイスに使われるストップビットの数を指定します。

表 7.14
通信ストップビット定数

定数	ストップビット数
ONESTOPBIT	1
ONE5STOPBITS	1.5
TWOSTOPBITS	2

CP_クリッピング機能

デバイスのクリッピング機能を指定します。

表 7.15
クリッピング機能定数

定数	意味
CP_NONE	出力のクリッピングなし。
CP_RECTANGLE	出力を矩形にクリップします。

CS_クラススタイル

WNDCLASSデータ構造体の ***Style*** フィールドで使うウィンドウクラススタイル定数です。この種の定数はビットごとの **OR** 演算子（|）で結合できます。

表 7.16
クラススタイル

定数	意味
CS_BYTEALIGNCLIENT	x 軸方向のバイト境界にクライアント領域を配置します。
CS_BYTEALIGNWINDOW	x 軸方向のバイト境界にウィンドウを配置します。
CS_CLASSDC	インスタンス専有のディスプレイコンテキストを，同じウィンドウクラスの他のインスタンスと共有します。
CS_DBLCLKS	マウスのダブルクリックメッセージをウィンドウが取得します。
CS_GLOBALCLASS	実行中のアプリケーションすべてがこのウィンドウクラスを使えます。
CS_HREDRAW	横の寸法が変わると，ウィンドウ全体が描き直されます。
CS_NOCLOSE	コントロールメニューのクローズオプションが使用不可になります。
CS_OWNDC	各ウィンドウインスタンスが，それぞれのディスプレイコンテキストを取得します。ウィンドウ当たり800バイトのメモリが使われます。
CS_PARENTDC	親ウィンドウのディスプレイコンテキストを使用します。
CS_SAVEBITS	現在表示されていないウィンドウの内容をビットマップに保管します。このビットマップによって内容が再表示できます。この定数の使用は必要最小限に止めます。
CS_VREDRAW	縦の寸法が変わると，ウィンドウ全体が描き直されます。

CTLCOLOR_コントロールカラーフラグ

描画されるコントロールまたはダイアログボックスの種類を示すフラグであり，***WM_CTLCOLOR*** メッセージで渡されます。

表 7.17
コントロールカラーフラグ

定数	意味
CTLCOLOR_BTN	ボタンコントロール。
CTLCOLOR_DLG	ダイアログボックス。
CTLCOLOR_EDIT	エディットコントロール。
CTLCOLOR_LISTBOX	リストボックス。
CTLCOLOR_MAX	最大コントロール。
CTLCOLOR_MSGBOX	メッセージボックス。
CTLCOLOR_SCROLLBAR	スクロールバー。
CTLCOLOR_STATIC	スタティックコントロール。

CW_USEDEFAULT 定数

CreateWindow または ***CreateWindowEx*** の引数として使われます。Windows に対し，作成するウィンドウにデフォルトのサイズまたは位置を割り当てるよう要求します。

Device capabilities（デバイス機能）

デバイスコンテキストの特定機能に対応した定数です。***GetDeviceCaps*** 関数で使われます。

表 7.18
デバイス機能定数

定数	意味
ASPECTX	相対的ピクセル幅。
ASPECTXY	対角線ピクセル長。
ASPECTY	相対的ピクセル高。
BITSPIXEL	ピクセル当たりのビット数。
CLIPCAPS	クリッピング機能。***CP***_クリッピング機能定数を返します。この節の「**CP_クリッピング機能**」を参照。
COLORRES	実際のカラー解像度（ピクセル当たりのビット数）。
CURVECAPS	曲線機能。***CC***_曲線作成機能定数を返します。この節の「**CC_曲線作成機能**」を参照。
DRIVERVERSION	デバイスドライババージョン。たとえば 0x100 は 1.0 になります。
HORZRES	幅（ピクセル単位）。
HORZSIZE	幅（mm 単位）。
LINECAPS	ライン機能。***LC***_ライン作成機能定数を返します。この節の「**LC_ライン作成機能**」を参照。
LOGPIXELSX	横 1 インチ当たりのピクセル数。
LOGPIXELSY	縦 1 インチ当たりのピクセル数。
NUMBRUSHES	デバイス所有のブラシ数。
NUMCOLORS	デバイスがサポートする色数。
NUMFONTS	デバイス所有のフォント数。
NUMMARKERS	デバイス所有のマーカ数。
NUMPENS	デバイス所有のペン数。
NUMRESERVED	パレットの予約エントリ数。
PDEVICESIZE	デバイス記述子に必要なサイズ。
PLANES	カラープレーン数。
POLYGONALCAPS	多角形作成機能。***PC***_多角形作成機能定数を返します。この節の「**PC_多角形作成機能**」を参照。
RASTERCAPS	ラスタ機能。***RC***_ラスタ機能定数を返します。この節の「**RC_ラスタ機能**」を参照。
SIZEPALETTE	物理パレットのエントリ数。

表 7.18
デバイス機能定数（つづき）

定数	意味
TECHNOLOGY	デバイスクラス。*DT_*デバイステクノロジー定数を返します。この節の「**DT_デバイステクノロジー**」を参照。
TEXTCAPS	テキスト機能。*TC_*テキスト機能定数を返します。この節の「**TC_テキスト機能**」を参照。
VERTRES	高さ（ピクセル単位）。
VERTSIZE	高さ（mm 単位）。

DIB_カラーテーブル識別子

デバイス独立ビットマップ用のカラーにアクセスする方法を規定します。この定数を用いる関数は，***CreateDIBitmap, CreateDIBPatternBrush, GetDIBits, SetDIBits, SetDIBitsToDevice, StretchDIBits*** です。

表 7.19
カラーテーブル識別子

定数	意味
DIB_PAL_COLORS	カラーテーブルを構成するのは，現在の論理パレットに配列された 16 ビットのインデックスです。
DIB_RGB_COLORS	カラーテーブルを構成するのは，RGB リテラル値です。

DLGC_ダイアログコード

Windows ではなく，アプリケーションに処理させるダイアログ入力の種類を指定します。***WM_GETDLGCODE*** メッセージからの戻り値です。

表 7.20
ダイアログコード

定数	意味
DLGC_DEFPUSHBUTTON	デフォルトのプッシュボタンメッセージ。
DLGC_HASSETSEL	***EM_SETSEL*** メッセージ。

表 7.20
ダイアログコード（つづき）

定数	意味
DLGC_RADIOBUTTON	ラジオボタンメッセージ。
DLGC_UNDEFPUSHBUTTON	プッシュボタンメッセージ。
DLGC_WANTALLKEYS	すべてのキーボード入力メッセージ。
DLGC_WANTARROWS	矢印キーメッセージ。
DLGC_WANTCHARS	*WM_CHAR* メッセージ。
DLGC_WANTMESSAGE	コントロールに渡されるすべてのキーボード入力メッセージ。
DLGC_WANTTAB	**TAB** キーメッセージ。

DLGWINDOWEXTRA ダイアログクラス定数

リソースからのダイアログにクラスを新たに定義するには，***WNDCLASS*** 構造体の ***cbWndExtra*** フィールドに ***DLGWINDOWEXTRA*** をセットします。

DM_DevMode フィールド選択定数

フィールド選択ビットとして使います。各定数が，***DEVMODE*** 構造体の ***dmFields*** フィールド内の特定ビットに対応しています。特定ビットがセットされていたら，そのビットに対応する構造体が初期化されたことを示します。

表 7.21
デバイスモードフィールド選択定数

定数	フィールド
DM_COLOR	*dmColor*
DM_COPIES	*dmCopies*
DM_DEFAULTSOURCE	*dmDefaultSource*
DM_DUPLEX	*dmDuplex*
DM_ORIENTATION	*dmOrientation*
DM_PAPERLENGTH	*dmPaperLength*
DM_PAPERSIZE	*dmPaperSize*
DM_PAPERWIDTH	*dmPaperWidth*
DM_PRINTQUALITY	*dmPrintQuality*
DM_SCALE	*dmScale*

DM_デバイスモード選択

ExtDeviceMode 関数の *Mode* 引数として，操作を指定します。*Mode* に以下の定数をいくつでも設定できます。

表 7.22
ExtDeviceMode モード定数

定数	**意味**
DM_COPY	プリンタドライバの設定を，***DevModeOutput*** で渡される ***DEVMODE*** レコードに書き込みます。
DM_MODIFY	プリンタドライバの設定を，***DevModeInput*** で渡される ***DEVMODE*** レコードの内容に変更します。
DM_PROMPT	プリンタドライバのセットアップ用ダイアログボックスを作成して，ユーザ指定のプリンタ設定にします。
DM_UPDATE	プリンタ環境と WIN.INI ファイルをプリンタの現設定に更新します。

DMBIN_デバイスモードビン選択定数

DEVMODE レコードの **dmDefaultSource** フィールドに使用します。デフォルトの給紙ビンを指定します。

DMIBIN_AUTO
DMIBIN_CASSETTE
DMIBIN_ENVELOPE
DMIBIN_ENVMANUAL
DMIBIN_LARGECAPACITY
DMIBIN_LARGEFMT
DMIBIN_LOWER
DMIBIN_MANUAL
DMIBIN_MIDDLE
DMIBIN_ONLYONE
DMIBIN_SMALLFMT
DMIBIN_TRACTOR
DMIBIN_UPPER

DMCOLOR_デバイスモードカラー定数

DEVMODE 構造体の *dmColor* フィールドに使用します。プリンタがカラー印刷をサポートしているかどうかを示します。

表 7.23
デバイスモードカラー定数

定数	意味
DMCOLOR_MONOCHROME	モノクロ印刷。
DMCOLOR_COLOR	カラー印刷。

DMDUP_デバイスモード両面印刷定数

DEVMODE 構造体の *dmDuplex* フィールドに使用します。両面印刷を指定します。

表 7.24
プリンタ両面印刷選択定数

定数	意味
DMDUP_HORIZONTAL	
DMDUP_SIMPLEX	片面印刷。
DMDUP_VERTICAL	

DMORIENT_デバイスモード印刷方向定数

DEVMODE 構造体の *dmOrientation* フィールドに使います。印刷方向を指定します。

表 7.25
プリンタ印刷方向定数

定数	意味
DMORIENT_PORTRAIT	ポートレイト。
DMORIENT_LANDSCAPE	ランドスケープ。

DMPAPER_デバイスモード用紙種別定数

DEVMODE レコード構造体の ***dmPaperSize*** フィールドに使用します。用紙サイズを指示します。

表 7.26
プリンタ用紙サイズ定数

定数	**幅**	**高さ**	**単位**
DMPAPER_10×14	10	14	インチ
DMPAPER_11×17	11	17	インチ
DMPAPER_A3	297	420	mm
DMPAPER_A4	210	297	mm
DMPAPER_A4SMALL	210	297	mm
DMPAPER_A5	148	210	mm
DMPAPER_B4	250	354	mm
DMPAPER_B5	182	257	mm
DMPAPER_CSHEET	Cサイズ		
DMPAPER_DSHEET	Dサイズ		
DMPAPER_ESHEET	Eサイズ		
DMPAPER_ENV_9	3 7/8	8 7/8	インチ
DMPAPER_ENV_10	4 1/8	9 1/2	インチ
DMPAPER_ENV_11	4 1/2	10 3/8	インチ
DMPAPER_ENV_12	4 3/4	11	インチ
DMPAPER_ENV_14	5	11 1/2	インチ
DMPAPER_EXECUTIVE	7 1/2	10	インチ
DMPAPER_FOLIO	8 1/2	13	インチ
DMPAPER_LEDGER	17	11	インチ
DMPAPER_LEGAL	8 1/2	14	インチ
DMPAPER_LETTER	8 1/2	11	インチ
DMPAPER_LETTERSMALL	8 1/2	11	インチ
DMPAPER_NOTE	8 1/2	11	インチ
DMPAPER_QUARTO	215	275	mm
DMPAPER_STATEMENT	5 1/2	8 1/2	インチ
DMPAPER_TABLOID	11	17	インチ

DMRES_デバイスモード解像度定数

DEVMODE 構造体の *dmPrintQuality* フィールドに使います。使用するプリンタの解像度を指示します。

表 7.27
プリンタ解像度定数

定数	意味
DMRES_DRAFT	ドラフト。
DMRES_LOW	低解像度。
DMRES_MEDIUM	中解像度。
EMRES_HIGH	高解像度。

DRIVE_ドライブタイプ

ディスクドライブのタイプを指定します。*GetDriveType* 関数から返される定数です。

表 7.28
ドライブタイプ定数

定数	意味
DRIVE_FIXED	ドライブは固定ドライブです。
DRIVE_REMOTE	ディスクはリモート（ネットワーク）ドライブです。
DRIVE_REMOVABLE	ドライブは交換可能です。

DS_ダイアログスタイル

CreateWindow と *CreateWindowEx* の両関数で作成するダイアログボックスのスタイルを指定します。

表 7.29
ダイアログスタイル

定数	意味
DS_ABSALIGN	オーナーウィンドウの左上隅ではなく画面の左上隅が，ダイアログボックスの配置基準点になります。
DS_LOCALEDIT	アプリケーションのデータセグメントのメモリを，ダイアログボックスの子コントロールに対応するデータ構造体に使用します。デフォルトではグローバルメモリが使用されます。このスタイルに限って，***EM_GETHANDLE*** と ***EM_SETHANDLE*** の両メッセージが使用できます。
DS_MODALFRAME	ダイアログボックスはモードつきダイアログボックスフレームを持ちます。このスタイルでは ***WS_CAPTION*** と ***WS_SYSMENU*** のスタイルが指定できます。
DS_NOIDLEMSG	オーナーに通常送られる ***WM_ENTERIDLE*** メッセージを除外します。
DS_SETFONT	コントロールの描画用に固有のフォントを設定します。ダイアログの作成には，***DLGTEMPLATE*** 構造体の拡張形式を使用します。拡張形式では，基本 ***DLGTEMPLATE*** 構造体のヘッダの後にフォント情報構造体が続きます。このスタイルを持つダイアログには，コントロールの作成前に ***WM_SETFONT*** メッセージが送られます。
DS_SYSMODAL	このスタイルのダイアログボックスが表示中は，すべての Windows アプリケーションが停止されます。

DT_デバイステクノロジー

使用するデバイスのタイプを表わします。

表 7.30
デバイスタイプ定数

定数	意味
DT_CHARSTREAM	文字ストリーム。
DT_DISPFILE	表示ファイル。
DT_METAFILE	メタファイル。
DT_PLOTTER	ベクトルプロッタ。
DT_RASCAMERA	ラスタカメラ。
DT_RASDISPLAY	ラスタ表示。
DT_RASPRINTER	ラスタプリンタ。

DT_テキスト描画書式フラグ

DrawText 関数の書式化オプションを指定します。

表 7.31 DrawText 書式化オプション

定数	意味
DT_BOTTOM	下端に位置合わせ。
DT_CALCRECT	テキストに合わせて矩形を計算し直しますが，テキストは実際には描画しません。
DT_CENTER	左右方向センタリング。
DT_EXPANDTABS	タブ文字の展開。
DT_EXTERNALLEADING	フォントの外部レディングを含むテキストの高さ。
DT_INTERNAL	フォントの内部レディングを含むテキストの高さ。
DT_LEFT	左寄せ。
DT_NOCLIP	クリッピングなしの描画。
DT_NOPREFIX	‘&’ などのプレフィクス文字を処理しません。
DT_RIGHT	右寄せ。
DT_SINGLELINE	単一行のみ。
DT_TABSTOP	タブストップ設定。
DT_TOP	上端に位置合わせ。
DT_VCENTER	上下方向センタリング。
DT_WORDBREAK	ワードラップ。

EN_エディットコントロール通知コード

エディットコントロールから *WM_COMMAND* メッセージで渡されるコードです。エディットコントロールで発生した事象をエディットコントロールのオーナーに知らせます。

表 7.32
エディットコントロール通知コード

定数	意味
EN_CHANGE	エディットコントロールのテキストが変更された。
EN_ERRSPACE	システムのメモリ不足。
EN_HSCROLL	水平スクロールバーのクリック。この通知は画面の更新前に行なわれます。
EN_KILLFOCUS	入力フォーカスの喪失。
EN_MAXTEXT	制限文字数の超過。
EN_SETFOCUS	入力フォーカスの取得。
EN_UPDATE	テキスト変更にともなうテキストの再表示を予告します。
EN_VSCROLL	垂直スクロールバーのクリック。この通知は画面の更新前に行なわれます。

ES_エディットコントロールスタイル

CreateWindow と ***CreateWindowEx*** の両関数で作成するエディットコントロールのスタイルを指定します。

表 7.33
エディットコントロールスタイル

定数	意味
ES_AUTOHSCROLL	行末で文字をタイプすると，エディットコントロール内のテキストは自動的に 10 文字分右にスクロールされます。**リターン**を押すとテキストが 0 位置に戻ります。
ES_AUTOVSCROLL	最終行のキャレットで**リターン**を押すと，テキストは自動的に 1 ページ分上方向にスクロールされます。
ES_CENTER	テキストのセンタリングを行ないます。このスタイルは ***ES_MULTILINE*** スタイルが指定されているときのみ使えます。
ES_LEFT	テキストを左寄せにします。このスタイルは ***ES_MULTILINE*** スタイルが指定されているときのみ使えます。
ES_LOWERCASE	入力文字をすべて小文字に変換します。
ES_MULTILINE	複数行を入力できるエディットコントロールです。このスタイルでは ***ES_AUTOVSCROLL*** スタイルだけが使用できます。***ES_AUTOVSCROLL*** スタイルを指定せずに，最終行のキャレットで**リターン**を押すと，ビープ音が鳴ります。また，***ES_AUTOHSCROLL*** スタイルを指定せずに語を入力すると，行末で自動的にワードラップが起こり改行されます。 ワードラップ位置はウィンドウサイズが変更されると変わります。スクロールバーのある複数行エディットコントロールでは，エディットコントロールが自分でスクロールバーメッセージを処理できますが，そうでない場合は上記のように自動的にスクロールされます。
ES_NOHIDESEL	入力フォーカス喪失後も選択項目が表示されたままです。デフォルトでは入力フォーカスの喪失とともに選択項目は非表示になります。
ES_OEMCONVERT	入力テキストを ANSI から OEM の文字セットへ変換し，後で ANSI に戻します。これで，***AnsiToOem*** 関数はエディットコントロールのテキストを正しく変換します。ファイル名を扱うエディットコントロールに便利なスタイルです。
ES_PASSWORD	入力文字をすべてアスタリスク（*）で表示します。***EM_SETPASSWORDCHAR*** メッセージで表示文字を変更できます。
ES_RIGHT	テキストを右に寄せます。このスタイルは ***ES_MULTILINE*** が指定されているときのみ使えます。
ES_UPPERCASE	入力文字すべてを大文字に変換します。

Escape comm（通信拡張）定数

通信デバイスが実行する機能コードを指定します。***EscapeCommFunction*** 関数の呼び出しで使用します。

表 7.34
EscapeCommFunction 定数

定数	意味
CLRDTR	DTR（data-terminal-ready）信号をクリアします。
CLRRTS	RTS（request-to-send）信号をクリアします。
RESETDEV	デバイスがリセット可能ならばリセットします。
SETDTR	DTR（data-terminal-ready）信号を送ります。
SETRTS	RTS（request-to-send）信号を送ります。
SETXOFF	XOFF 文字の受信をシミュレートします。
SETXON	XON 文字の受信をシミュレートします。

ETO_ExtTextOut オプション

ExtTextOut 関数で使うバックグラウンド描画方法を指定します。

表 7.35
ExtTextOut オプション

定数	意味
ETO_CLIPPED	テキストは矩形領域に合わせてクリップされます。
ETO_OPAQUE	バックグラウンド色で矩形領域が塗りつぶされます。

EV_通信イベント定数

通信デバイスのイベントマスクへのポインタであり，***SetCommEventMask*** 関数の戻り値です。以下の各定数は，対応するマスクビットに1がセットされたことを示します。

表 7.36
SetCommEventMask 戻り値

定数	意味
EV_BREAK	ブレークの受信。
EV_CTS	CTS（clear-to-send）信号による状態の変化。
EV_DSR	DSR（data-set-ready）信号による状態の変化。
EV_ERR	ラインステータスエラーの発生。
EV_PERR	プリンタエラーの発生。
EV_RING	リングインジケータの検出。
EV_RLSD	RLSD（Receive-line-signal-detect）信号による状態の変化。
EV_RXCHAR	文字の受信。
EV_RXFLAG	デバイスのコントロールブロックで指定したイベント文字を受信。
EV_TXEMPTY	送信キューが空。

FLOODFILL（塗りつぶし）スタイルフラグ

ExtFloodFill 関数で使う塗りつぶし種別を指定します。

表 7.37
塗りつぶしスタイルフラグ

定数	意味
FLOODFILLBORDER	指定色で囲まれた領域が塗りつぶされます。この種の塗りつぶしは，***FloodFill*** 関数によって行なわれます。
FLOODFILLSURFACE	指定色を含む領域が塗りつぶされます。色が複数で形が複雑な面を塗りつぶすのに最適です。

FF_フォントファミリーフラグ

CreateFont 関数で作成するフォントの文字セットを指定します。

表 7.38
CreateFont 文字セットオプション

定数	意味
FF_DECORATIVE	オールドイングリッシュなどの特殊フォント。
FF_DONTCARE	指定なし。
FF_MODERN	パイカ，エリート，クーリエなど，固定ピッチの幅でセリフ体またはサンセリフ体フォント。
FF_ROMAN	タイムズローマン，センチュリースクールブックなど，可変ピッチのセリフ体フォント。
FF_SCRIPT	スクリプトなどの手書きに似たフォント。
FF_SWISS	ヘルベチカ，スイスなど，可変ピッチのサンセリフ体フォント。

Font character set（フォント文字セット）フラグ

CreateFont 関数で作成するフォントの文字セットを指定します。

ANSI_CHARSET
SYMBOL_CHARSET
HANGEUL_CHARSET
SHIFTJIS_CHARSET
OEM_CHARSET

Font output quality（フォント出力品質）フラグ

CreateFont 関数で作成するフォントの出力品質を指定します。

表 7.39
CreateFont 出力品質オプション

定数	意味
DEFAULT_QUALITY	普通の出力品質。
DRAFT_QUALITY	少し改良した出力品質。
PROOF_QUALITY	高品質出力。論理フォントの属性とのマッチングよりも文字質を優先します。

Font pitch（フォントピッチ）フラグ

CreateFont 関数で作成するフォントのピッチを指定します。

DEFAULT_PITCH
FIXED_PITCH
VARIABLE_PITCH

FW_フォントウェイトフラグ

CreateFont 関数で作成するフォントの太さを指定します。以下の一覧は，最も細いもの（***FW_THIN***）から最も太いもの（***FW_HEAVY***）へと順に並べてあります。

FW_DONTCARE
FW_THIN
FW_EXTRALIGHT, FW_ULTRALIGHT
FW_NORMAL, FW_REGULAR
FW_MEDIUM
FW_SEMIBOLD, FW_DEMIBOLD
FW_BOLD, FW_BLACK
FW_EXTRABOLD, FW_ULTRABOLD
FW_HEAVY

GCL_クラスフィールドオフセット

GetClassLong と *SetClassLong* 関数で取得変更するウィンドウクラス情報のバイトオフセットを指定します。

表 7.40
GetClassLong または SetClassLong のクラスフィールドオフセット

定数	意味
GCL_MENUNAME	メニュー名へのポインタ（***SetClassLong***）のみ。
GCL_WNDPROC	ウィンドウ関数へのポインタ。

GCW_クラスフィールドオフセット

GetClassWord と *SetClassWord* 関数で取得，変更するウィンドウクラス情報のバイトオフセットを指定します。

表 7.41
GetClassWord または SetClassWord のクラスフィールドオフセット

定数	意味
GCW_CBCLSEXTRA	クラス情報の後に割り当てるバイト数の取得または設定。
GCW_CBWNDEXTRA	ウィンドウ情報の後に割り当てるバイト数の取得または設定。
GCW_HBRBACKGROUND	バックグラウンドブラシへのハンドル。
GCW_HCURSOR	カーソルへのハンドル。
GCW_HICON	アイコンへのハンドル。
GCW_HMODULE	モジュールへのハンドル（***GetClassWord*** のみ）。
GCW_STYLE	ウィンドウクラスのスタイルビット。

GMEM_グローバルメモリフラグ

GlobalAlloc と *GlobalReAlloc* の両関数で作成するグローバルメモリブロックの特性を指定します。*GlobalFlags* と *GetFreeSpace* の両関数でも使用します。

表 7.42
グローバルメモリフラグ

定数	意味
GMEM_DDESHARE	ダイナミックデータエクスチェンジ（DDE）プロトコルにより多数のアプリケーションでメモリブロックを共有できます。割り当てたアプリケーションの終了とともに，ブロックは廃棄されます。
GMEM_DISCARDABLE	メモリブロックは廃棄可能です。***GMEM_MOVEABLE*** と併用します。
GMEM_DISCARDED	メモリブロックは廃棄済みです。***GlobalFlags*** にのみ使用します。
GMEM_FIXED	メモリブロックの位置は，メモリの一定箇所に固定されます。
GMEM_LOCKCOUNT	***GlobalFlags*** からの戻り値の下位バイトと組み合わされて，メモリブロックのロックカウントを返します。***GlobalFlags*** にのみ使用します。
GMEM_LOWER	***GMEM_NOT_BANKED*** と同じです。
GMEM_MODIFY	このフラグを指定すると，グローバルメモリフラグが変更されます。***GlobalReAlloc*** にのみ使用します。
GMEM_MOVEABLE	メモリブロックの位置は，メモリの一定箇所に固定されません。
GMEM_NOCOMPACT	他メモリブロックを圧縮，廃棄せずに，メモリブロックを割り当てます。
GMEM_NODISCARD	他メモリブロックを廃棄せずに，メモリブロックを割り当てます。
GMEM_NOT_BANKED	メモリブロックの割り当ては，バンクアウトされていないメモリに行なわれます。
GMEM_NOTIFY	メモリが廃棄されると，通知ルーチンが呼び出されます。
GMEM_SHARE	***GMEM_DDESHARE*** と同じです。
GMEM_ZEROINIT	メモリブロックの内容を 0 に初期化します。

GW_ウィンドウ取得定数

GetNextWindow と ***GetWindow*** の両関数の呼び出しで，指定したウィンドウと検索対象のウィンドウとの関係を指定します。***GetNextWindow*** 関数が受けつける定数は，***GW_HWNDNEXT*** と ***GW_HWNDPREV*** のみです。

表 7.43
ウィンドウ取得定数

定数	意味
GW_CHILD	最初の子ウィンドウ。
GW_HWNDFIRST	子ウィンドウにとっての最初の兄弟ウィンドウ。
GW_HWNDLAST	子ウィンドウにとっての最後の兄弟ウィンドウ。
GW_HWNDNEXT	ウィンドウマネージャリスト上の次のウィンドウ。
GW_HWNDPREV	ウィンドウマネージャリスト上の前のウィンドウ。
GW_OWNER	ウィンドウのオーナー。

GWL_ウィンドウフィールドオフセット

GetWindowLong または ***SetWindowLong*** で取得，変更するウィンドウ属性のバイトオフセットを指定します。

表 7.44
GetWindowLong または SetWindowLong のウィンドウフィールドオフセット

定数	意味
GWL_EXSTYLE	拡張ウィンドウスタイル。
GWL_STYLE	ウィンドウスタイル。
GWL_WNDPROC	ウィンドウ関数へのポインタ。

GWW_ウィンドウフィールドオフセット

GetWindowWord または ***SetWindowWord*** で取得，変更するウィンドウ情報のバイトオフセットを指定します。

表 7.45
GetWindowWord または SetWindowWord のウィンドウフィールドオフセット

定数	意味
GWW_HINSTANCE	ウィンドウを所有するモジュールのインスタンス識別子。
GWW_HWNDPARENT	親ウィンドウ（***GetWindowWord*** にのみ使用)。
GWW_ID	子ウィンドウのコントロール ID。

HELP_ヘルプコマンド

アプリケーションが要求するヘルプ種別を，Windows のヘルプ機能に指示します。***WinHelp*** 関数の呼び出しに使う定数です。

表 7.46
ヘルプコマンド定数

定数	意味
HELP_CONTEXT	*Data* 引数で指定したコンテキストに関するヘルプ。
HELP_HELPONHELP	ヘルプ機能の使い方についてのヘルプ。
HELP_INDEX	ヘルプ索引の表示。
HELP_KEY	*Data* 引数で指定したキーワードに関するヘルプ。
HELP_MULTIKEY	代替キーワードテーブルのキーワードに関するヘルプ。
HELP_QUIT	ヘルプ機能の終了。
HELP_SETINDEX	*Data* 引数で指定したヘルプ索引を表示します。ただし，この *Data* 引数は *HelpFile* 引数で指定されたファイルの引数です。

HS_ハッチスタイル

ブラシのハッチスタイルを指定します。***CreateHatchBrush*** 関数で使用される定数です。

表 7.47
ハッチスタイル定数

定数	意味
HS_BDIAGONAL	＼＼＼＼＼
HS_CROSS	＋＋＋＋＋
HS_DIAGCROSS	×××××
HS_FDIAGONAL	／／／／／
HS_HORIZONTAL	―――――
HS_VERTICAL	｜｜｜｜｜

HT ヒットテストコード

カーソル位置をウィンドウに対する相対位置として指示するコードです。***WM_MOUSEACTIVATE, WM_SETCURSOR, WM_NCHITTEST*** など，カーソルの配置や移動を伴なう多くのメッセージで渡されます。

表 7.48
ヒットテストコード

定数	意味
HTBOTTOM	ウィンドウの下辺枠。
HTBOTTOMLEFT	ウィンドウの左下隅。
HTBOTTOMRIGHT	ウィンドウの右下隅。
HTCAPTION	キャプション領域。
HTCLIENT	クライアント領域。
HTERROR	***HTNOWHERE*** 定数と同じだが，ビープ音つき。
HTGROWBOX	サイズボックス。
HTHSCROLL	水平スクロールバー。
HTLEFT	ウィンドウの左辺枠。
HTMENU	メニュー領域。
HTNOWHERE	画面バックグラウンド，またはウィンドウ間の境界線。
HTREDUCE	最小化ボタン。
HTRIGHT	ウィンドウの右辺枠。
HTSIZE	***HTGROWBOX*** に同じ。
HTSYSMENU	コントロールメニューボックス。
HTTOP	ウィンドウの上辺枠。
HTTOPLEFT	ウィンドウの左上隅。
HTTOPRIGHT	ウィンドウの右上隅。
HTTRANSPARENT	別のウィンドウで覆われている現在のウィンドウ。
HTVSCROLL	垂直スクロールバー。
HTZOOM	最大化ボタン。

ID ダイアログボックスコマンド ID

MessageBox 関数の戻り値です。メッセージボックスの選択を示します。

表 7.49
ダイアログボックスコマンド ID 定数

定数	意味
IDABORT	「中止」ボタン選択。
IDCANCEL	「取消」ボタン選択。
IDIGNORE	「無視」ボタン選択。
IDNO	「いいえ」ボタン選択。
IDOK	「了解」ボタン選択。
IDRETRY	「再試行」ボタン選択。
IDYES	「はい」ボタン選択。

IDC_標準カーソル ID

Windows が定義するストックカーソルを識別します。***LoadCursor*** 関数でロードします。

表 7.50
標準カーソル ID 定数

定数	意味
IDC_ARROW	矢印型。
IDC_CROSS	十字型。
IDC_IBEAM	テキスト用の I 字型。
IDC_ICON	空のアイコン型（同心正方形)。
IDC_SIZE	4 方向矢印型（上下左右向き)。
IDC_SIZENESW	2 方向矢印型（右斜め向き)。
IDC_SIZENS	2 方向矢印型（上下向き)。
IDC_SIZENWSE	2 方向矢印型（左斜め向き)。
IDC_SIZEWE	2 方向矢印型（左右向き)。
IDC_UPARROW	上方矢印型。
IDC_WAIT	砂時計型。

IDI_標準アイコン ID

Windows が定義するストックアイコンを識別します。***LoadIcon*** 関数でロードします。

表 7.51
標準アイコン ID 定数

定数	意味
IDI_APPLICATION	デフォルト。
IDI_ASTERISK	情報メッセージ用の '*'。
IDI_EXCLAMATION	警告用の '!'。
IDI_HAND	重大な警告用の「手」のアイコン。
IDI_QUESTION	問い合わせメッセージ用の '?'。

IE_通信オープンエラーフラグ

OpenComm 関数が返す負の値で，通信デバイスのオープンエラーを示します。

表 7.52
OpenComm エラーフラグ

定数	意味
IE_BADID	無効な ID。
IE_BAUDRATE	サポートされていない通信速度。
IE_BYTESIZE	無効なバイトサイズ。
IE_DEFAULT	デフォルト引数のエラー。
IE_HARDWARE	ハードウェア不在。
IE_MEMORY	キューの割り当て不能。
IE_NOPEN	デバイスがオープンできない。
IE_OPEN	デバイスがすでにオープンされている。

LB_リストボックス戻り値

LB_ADDSTRING などの，Windows リストボックスメッセージから返される値です。負の値はエラーを示します。

表 7.53
リストボックスメッセージの戻り値

定数	意味
LB_ERR	エラー発生のため処理は行なわれなかった。
LB_ERRSPACE	リストボックスのスペース不足のため処理は行なわれなかった。
LB_OKAY	エラーなし。

LBN_リストボックス通知コード

リストボックスのコントロールが生成した *WM_COMMAND* メッセージで渡される通知コードです。リストボックスで発生した事象を，親ウィンドウに知らせます。

表 7.54
リストボックス通知コード

定数	意味
LBN_DBLCLK	マウスによる項目のダブルクリック。
LBN_ERRSPACE	システムのメモリ不足。
LBN_KILLFOCUS	入力フォーカスの喪失。
LBN_SELCHANGE	別項目の選択。
LBN_SETFOCUS	入力フォーカスの取得。

LBS_リストボックススタイル

CreateWindow, *CreateWindowEx* の両関数で作成するリストボックスのスタイルを指定します。

表 7.55
リストボックススタイル

定数	意味
LBS_EXTENDEDSEL	**SHIFT** キーとマウスの併用，または他の特殊なキーの組み合わせによって複数項目の選択ができます。
LBS_HASSTRINGS	***LBS_OWNERDRAWFIXED*** または ***LBS_OWNERDRAWVARIABLE*** スタイルと併用できます。このスタイルのリストボックスは，項目に文字列を使用します。文字列を管理するのはシステムであり，取得するには ***LB_GETTEXT*** メッセージを使用します。
LBS_MULTICOLUMN	複数のカラムがあり，水平にスクロールすることができます。カラム幅の設定には，***LB_SETCOLUMNWIDTH*** メッセージを用います。
LBS_MULTIPLESEL	マウスで複数項目を選択できるリストボックスです。項目をクリックまたはダブルクリックするたびに，選択と選択解除が切り換わります。
LBS_NOINTEGRALHEIGHT	実際のリストボックスのサイズが作成時に指定したサイズであることを示します。通常は作成時のサイズを変更して，項目を一部隠すことができます。
LBS_NOREDRAW	変更が加えられても再描画されないリストボックスです。***WM_SETREDRAW*** メッセージでこのスタイルを動的にセットまたはリセットできます。
LBS_NOTIFY	項目のクリックまたはダブルクリックのたびに，入力メッセージを親ウィンドウに送ります。
LBS_OWNERDRAWFIXED	リストボックスのオーナーが，リストボックスの内容を描画します。項目の高さは一定になります。
LBS_OWNERDRAWVARIABLE	リストボックスのオーナーが，リストボックスの内容を描画します。各項目を別々の高さにできます。
LBS_SORT	項目をアルファベット順に並び換えます。***LBS_OWNERDRAWFIXED*** と ***LBS_OWNERDRAWVARIABLE*** とでは，リストボックスの並び順が異なる場合があります。
LBS_STANDARD	この定数は ***LBS_NOTIFY*** と ***LBS_SORT*** との合成です。リストボックスは枠で囲われます。
LBS_USETABSTOPS	項目にタブ文字を展開できます。デフォルトでは，項目の左端から 32 ダイアログ単位ごとにタブストップが設けられます。ダイアログ単位の 4 倍がダイアログの基本幅単位で，***GetDialogBaseUnits*** 関数で得られます。
LBS_WANTKEYBOARDINPUT	入力フォーカスがあるリストボックスでキーが押されると，***WM_VKEYTOITEM*** か ***WM_CHARTOITEM*** のメッセージがオーナーに送られます。

LC_ライン作成機能

デバイスによるライン作成機能を表わす定数です。

表 7.56
ライン作成機能定数

定数	意味
LC_NONE	ラインのサポートなし。
LC_INTERIORS	インテリア。
LC_MARKER	マーカ。
LC_POLYLINE	折れ線。
LC_POLYMARKER	多角マーカ。
LC_STYLED	スタイル線。
LC_WIDE	太線。
LC_WIDESTYLED	太いスタイル線。

LF_FACESIZE 論理フォントサイズ定数

LOGFONT 構造体の ***lfFaceName*** フィールドにフォント名を格納するのに使われるバイト数を示します。現在設定されているバイト数は 32 です。

LMEM_ローカルメモリフラグ

LocalAlloc と ***LocalReAlloc*** の両関数によって作成されるローカルメモリブロックの特性を指定します。***LMEM_DISCARDABLE, LMEM_DISCARDED, LMEM_LOCKCOUNT*** は，***LocalFlags*** 関数で使用されます。

表 7.57
ローカルメモリフラグ

定数	意味
LMEM_DISCARDABLE	メモリブロックは廃棄可能です。必ず ***LMEM_MOVEABLE*** と併用します。
LMEM_DISCARDED	メモリブロックは廃棄済みです。**LocalFlags** のみに使用します。
LMEM_LOCKCOUNT	**LocalFlags** からの戻り値の下位バイトと組み合わされると，メモリブロックのロックカウントを返します。**LocalFlags** にのみ使用します。
LMEM_FIXED	メモリブロックは特定のメモリ位置に固定されます。
LMEM_MODIFY	***LMEM_DISCARDABLE*** フラグを変更します。
LMEM_MOVEABLE	メモリブロックはメモリ内を移動できます。
LMEM_NOCOMPACT	メモリブロックを割り当てるとき，他メモリブロックを圧縮または廃棄しません。
LMEM_NODISCARD	メモリブロックを割り当てるとき，他メモリブロックは廃棄されません。
LMEM_ZEROINIT	メモリブロックの内容を 0 に初期化します。

LPTx 定数

LPTx は ***DCB Id*** フィールドのビットマスクです。このビットマスクがセットされていたら，デバイスは LPT（パラレルポート）です。

MA_マウス起動コード

このコードは ***WM_MOUSEACTIVATE*** メッセージから返されるときに，ウィンドウをアクティブにするか，マウスイベントの廃棄を行なうかどうかを指示します。

表 7.58
マウス起動コード

定数	意味
MA_ACTIVATE	ウィンドウをアクティブにします。
MA_ACTIVATEANDEAT	ウィンドウをアクティブにして，マウスイベントを廃棄します。
MA_NOACTIVATE	ウィンドウをアクティブにしません。

MB_メッセージボックスフラグ

MessageBox 関数で作成されるメッセージボックスの特性を指定します。作成するスタイルに合わせて，フラグを組み合わせて用います。

表 7.59
メッセージボックスフラグ

定数	**意味**
MB_ABORTRETRYIGNORE	「Abort」，「Retry，「Ignore」の各ボタンのみ。
MB_APPLMODAL	モードつきメッセージボックスの作成(デフォルト指定)。
MB_DEFBUTTON1	最初のボタンをデフォルトとする（デフォルト指定)。
MB_DEFBUTTON2	2番目のボタンをデフォルトとする。
MB_DEFBUTTON3	3番目のボタンをデフォルトとする。
MB_ICONASTERISK	***MB_ICONINFORMATION*** に同じ。
MB_ICONEXCLAMATION	'!' アイコン。
MB_ICONHAND	***MB_ICONSTOP*** に同じ。
MB_ICONINFORMATION	'i' アイコン。
MB_ICONQUESTION	'?' アイコン。
MB_ICONSTOP	「手」のアイコン。
MB_OK	「OK」ボタンのみ。
MB_OKCANCEL	「OK」と「Cancel」の各ボタンのみ。
MB_RETRYCANCEL	「Retry」と「Cancel」の各ボタンのみ。
MB_SYSTEMMODAL	Windowsを停止するモードつきメッセージボックスを作成します。ダメージが予想される状況に使用します。
MB_TASKMODAL	親ウィンドウが使えないときに，このフラグを使います。親ウィンドウの引数を0にするとアプリケーションの最上位ウィンドウがすべて停止されます。
MB_YESNO	「Yes」と「No」の各ボタンのみ。
MB_YESNOCANCEL	「Yes」，「No」，「Cancel」の各ボタンのみ。

MB_ 定数群に対し何種類かのビットマスクが定義されています。

表 7.60
メッセージボックスフラグマスク

マスク	**マスクされる定数**
MB_DEFMASK	***MB_DEFBUTTON1, MB_DEFBUTTON2, MB_DEFBUTTON3***
MB_ICONMASK	***MB_ICONASTERISK, MB_ICONEXCLAMATION, MB_ICONHAND, MB_ICONINFORMATION, MB_ICONQUESTION, MB_ICONSTOP***
MB_MODEMASK	***MB_APPLMODAL, MB_SYSTEMMODAL, MB_TASKMODAL***
MB_TYPEMASK	***MB_ABORTRETRYIGNORE, MB_OK, MB_OKCANCEL, MB_RETRYCANCEL, MB_YESNO, MB_YESNOCANCEL***

META_メタファイルコード

特定のGDI関数に対応します。たとえば，*META_ARC*の値は*Arc*GDI関数の数値インデックスです。この定数と，それに対応する関数のインデックス値はWindowsのメタファイルに格納できます。このメタファイルはGDIコマンドのリストになっており，グラフィックス出力を作成するためのものです。

META_ANIMATEPALETTE
META_ARC
META_BITBLT
META_CHORD
META_CREATEBITMAP
META_CREATEBITMAPINDIRECT
META_CREATEBRUSH
META_CREATEBRUSHINDIRECT
META_CREATEFONTINDIRECT
META_CREATEPALETTE
META_CREATEPATTERNBRUSH
META_CREATEPENINDIRECT
META_CREATEREGION
META_DELETEOBJECT
META_DIBBITBLT
META_DIBCREATEPATTERNBRUSH
META_DIBSTRETCHBLT
META_DRAWTEXT
META_ELLIPSE
META_ESCAPE
META_EXCLUDECLIPRECT
META_EXTTEXTOUT
META_FILLREGION
META_FLOODFILL
META_FRAMEREGION
META_INTERSECTCLIPRECT
META_INVERTREGION
META_LINETO
META_MOVETO
META_OFFSETCLIPRGN
META_OFFSETVIEWPORTORG
META_OFFSETWINDOWORG
META_PAINTREGION
META_PATBLT
META_PIE
META_POLYGON
META_POLYLINE
META_POLYPOLYGON
META_REALIZEPALETTE
META_RECTANGLE
META_RESIZEPALETTE
META_RESTOREDC
META_ROUNDRECT
META_SAVEDC
META_SCALEVIEWPORTEXT
META_SCALEWINDOWEXT
META_SELECTCLIPREGION
META_SELECTOBJECT
META_SELECTPALETTE
META_SETBKCOLOR
META_SETBKMODE
META_SETDIBTODEV
META_SETMAPMODE
META_SETMAPPERFLAGS

META_SETPALENTRIES	*META_SETTEXTCOLOR*
META_SETPIXEL	*META_SETTEXTJUSTIFICATION*
META_SETPOLYFILLMODE	*META_SETVIEWPORTEXT*
META_SETRELABS	*META_SETVIEWPORTORG*
META_SETROP2	*META_SETWINDOWEXT*
META_SETSTRETCHBLTMODE	*META_SETWINDOWORG*
META_SETTEXTALIGN	*META_STRETCHBLT*
META_SETTEXTCHAREXTRA	*META_TEXTOUT*

MF_メニューフラグ

メニュー関数と *WM_MENUSELECT* メッセージでフラグとして使います。

表 7.61
メニューフラグ

定数	意味
MF_BITMAP	メニュー項目はビットマップであり，文字列ではありません。
MF_BYCOMMAND	メニュー項目を ID で指定します。
MF_BYPOSITION	メニュー項目を位置で指定します。最初の項目位置は 0 です。
MF_CHECKED	メニュー項目の横にチェックマークを表示します。
MF_DISABLED	メニュー項目を使用不可にします。
MF_ENABLED	メニュー項目を使用可にします。
MF_GRAYED	メニュー項目を淡色表示にして使用不可を示します。
MF_HELP	ヘルプメニュー項目を示します。
MF_HILITE	メニュー項目のハイライトをオンにします。
MF_MENUBARBREAK	ポップアップメニュー項目を別のカラムに移してバーで分けます。
MF_MENUBREAK	ポップアップメニュー項目またはメニューバー項目をそれぞれ別のカラムまたは行に移します。
MF_MOUSESELECT	マウスで項目が選択されたことを示します。
MF_OWNERDRAW	オーナー描画項目であることを示します。
MF_POPUP	メニュー項目にサブ項目があることを示します。
MF_SEPARATOR	メニューに水平区切りバーを挿入します。
MF_STRING	メニュー項目が文字列であることを示します。
MF_SYSMENU	コントロールメニューの項目であることを示します。
MF_UNCHECKED	チェックマークがあれば除去します。
MF_UNHILITE	メニュー項目のハイライトをオフにします。

MK_キー状態マスク

マウスクリックのメッセージで渡され，特定キーとマウスボタンの状態を知らせます。組み合わせて用いることもできます。

表 7.62
キー状態マスク

定数	意味
MK_CONTROL	***CTRL*** キーが押されている。
MK_LBUTTON	左のマウスボタンが押されている。
MK_MBUTTON	中央のマウスボタンが押されている。
MK_RBUTTON	右のマウスボタンが押されている。
MK_SHIFT	***SHIFT*** キーが押されている。

M
Cons

MM_マッピングモード

論理単位からデバイス単位への変換と座標の変換を行なうための，デバイスコンテキストのマッピングモードを指定します。***GetMapMode*** と ***SetMapMode*** の両関数に使用します。

表 7.63
マッピングモード

定数	意味
MM_ANISOTROPIC	単位と座標軸の決定は任意に行なわれます。
MM_HIENGLISH	1 論理単位は 0.001 インチです。X 軸の正方向は右，Y 軸の正方向は上です。
MM_HIMETRIC	1 論理単位は 0.01 ミリメートルです。X 軸の正方向は右，Y 軸の正方向は上です。
MM_ISOTROPIC	X 軸の 1 論理単位は Y 軸の 1 論理単位に等しくなります。
MM_LOENGLISH	1 論理単位は 0.01 インチです。X 軸の正方向は右，Y 軸の正方向は上です。
MM_LOMETRIC	1 論理単位は 0.1 ミリメートルです。X 軸の正方向は右，Y 軸の正方向は上です。
MM_TEXT	1 論理単位は 1 ピクセルです。X 軸の正方向は右, Y 軸の正方向は下です。
MM_TWIPS	1 論理単位は 1/1440 インチです。X 軸の正方向は右，Y 軸の正方向は上です。

MSGF_フィルタ処理コード

メッセージフィルタ関数である ***WH_MSGFILTER*** と ***WH_SYSMSGFILTER*** の *Code* 引数で渡されて，処理対象のメッセージタイプを指定します。

表 7.64
フィルタフックによって処理されるメッセージ

定数	処理メッセージ *WH_MSGFILTER*	処理メッセージ *WH_SYSMSGFILTER*
MSGF_DIALOGBOX	メッセージボックスとダイアログボックス	ダイアログボックスのみ。
MSGF_MENU	キーボードとマウス	キーボードとマウス。
MSGF_MESSAGEBOX		メッセージボックスのみ。

OBJ_GDI オブジェクト型定数

EnumObjects 関数の呼び出しで列挙する GDI オブジェクトの型を指定します。ペンかブラシのいずれかです。***EnumObjects*** の *ObjectType* 引数で渡してください。

表 7.65
オブジェクト列挙定数

定数	意味
OBJ_PEN	ペンを列挙します。
OBJ_BRUSH	ブラシを列挙します。

OBM_定義済みビットマップ

Windows による定義済みのビットマップを指定します。プログラマはこれらを ***LoadBitmap*** 関数で使用できます。***OBM_OLD*** で始まる定数は，Windows の 3.0 より前のバージョンに使用されるビットマップを指します。

OBM_BTNCORNERS
OBM_BTSIZE
OBM_CHECK
OBM_CHECKBOXES
OBM_CLOSE
OBM_COMBO
OBM_DNARROW
OBM_DNARROWD
OBM_LFARROW
OBM_LFARROWD
OBM_MNARROW
OBM_OLD_CLOSE
OBM_OLD_DNARROW
OBM_OLD_LFARROW
OBM_OLD_REDUCE
OBM_OLD_RESTORE
OBM_OLD_RGARROW
OBM_OLD_UPARROW
OBM_OLD_ZOOM
OBM_REDUCE
OBM_REDUCED
OBM_RESTORE
OBM_RESTORED
OBM_RGARROW
OBM_RGARROWD
OBM_SIZE
OBM_UPARROW
OBM_UPARROWD
OBM_ZOOM
OBM_ZOOMD

ODA_オーナー描画処理

オーナー描画コントロールの処理を指定した定数で，***DRAWITEMSTRUCT*** 構造体の ***itemAction*** フィールドに使用します。このフィールドに，これらの定数を複数使うことができます。

表 7.66
オーナー描画コントロール処理定数

定数	意味
ODA_DRAWENTIRE	コントロール全体の描画。
ODA_FOCUS	フォーカスの喪失または取得。
ODA_SELECT	選択状態の変化。

ODS_オーナー描画状態

オーナー描画コントロールの描画後の状態を指定した定数で，***DRAWITEMSTRUCT*** 構造体の ***itemState*** フィールドに使用します。このフィールドに，これらの定数を複数使うことができます。

表 7.67
オーナー描画コントロール状態定数

定数	意味
ODS_CHECKED	メニュー項目をチェックします。
ODS_DISABLED	項目を無効にします。
ODS_FOCUS	項目に入力フォーカスを与えます。
ODS_GRAYED	メニュー項目を淡色表示にします。
ODS_SELECTED	項目を選択します。

ODT_オーナー描画コントロール種別

オーナー描画コントロールの種別を指定します。***COMPAREITEMSTRUCT***, ***DRAWITEMSTRUCT***, ***MEASUREITEMSTRUCT*** の各構造体の ***Ctltype*** フィールドに使用します。

表 7.68
オーナー描画コントロール種別定数

定数	意味
ODT_MENU	オーナー描画メニュー。
ODT_LISTBOX	オーナー描画リストボックス。
ODT_COMBOBOX	オーナー描画コンボボックス。
ODT_BUTTON	オーナー描画ボタン。

OF_ファイルオープン定数

OpenFile または ***_lopen*** 関数の呼び出しで組み合わせて渡すと，ファイルのオープン処理を指定できます。

表 7.69
ファイルオープン定数

定数	意味
OF_CANCEL	***OF_PROMPT*** 定数で作成したダイアログボックスに「取消」ボタンを加えます。「取消」を押すと，***OpenFile*** は「ファイルが見つからない」というエラーを返します。この定数は ***OpenFile*** のみに使用します。
OF_CREATE	新ファイルのオープンまたは既存ファイルのクリアを行ないます。***OpenFile*** のみに使用します。
OF_DELETE	ファイルを削除します。***OpenFile*** のみに使用します。
OF_EXIST	指定ファイルの存在を確認します。***OpenFile*** のみに使用します。
OF_PARSE	*ReOpenBuff* 引数で渡された ***OFSTRUCT*** 構造体へのデータの格納のみを行ないます。***OpenFile*** のみに使用します。
OF_PROMPT	該当ファイルがない場合に，「ファイルが見つかりません」というダイアログボックスを表示します。***OpenFile*** のみに使用します。
OF_READ	ファイルを読み込み専用としてオープンします。
OF_READWRITE	ファイルを読み書き両用としてオープンします。
OF_REOPEN	*ReOpenBuff* 引数で渡された ***OFSTRUCT*** 構造体の情報を使ってファイルをオープンします。***OpenFile*** のみに使用します。
OF_SHARE_COMPAT	オープンされるファイルは，他のプロセスから何回でもオープンすることができます。
OF_SHARE_DENY_NONE	オープンされるファイルは，他のプロセスで読み書きできます。
OF_SHARE_DENY_READ	オープンされるファイルを，他のプロセスから読み込めません。
OF_SHARE_DENY_WRITE	オープンされるファイルは，他のプロセスから書き込めません。
OF_SHARE_EXCLUSIVE	オープンされるファイルは，他のプロセスから読み書きができません。
OF_VERIFY	ファイルの年月日と時間が，前回オープン時のものと同じかどうかを確認します。
OF_WRITE	ファイルを書き込み専用としてオープンします。

OUT_フォント出力精度フラグ

CreateFont 関数で作成するフォントの出力精度を指定します。

OUT_DEFAULT_PRECIS	*OUT_CHARACTER_PRECIS*
OUT_STRING_PRECIS	*OUT_STROKE_PRECIS*

PC_パレットエントリフラグ

パレット情報を指定します。***PALETTEENTRY*** 構造体の ***peFlags*** フィールドで使用され，***AnimatePalette***, ***GetPaletteEntries***, ***GetSystemPaletteEntries***, ***SetPaletteEntries*** の各関数の呼び出しで渡されます。

表 7.70
パレットエントリフラグ

定数	意味
PC_EXPLICIT	論理パレットエントリの下位ワードは，物理パレットインデックスを指定します。
PC_NOCOLLAPSE	既存のパレット色とは照合されず，別の色としてシステムパレットに加えられます。
PC_RESERVED	パレットアニメーション用の色であり，頻繁に変化します。

PC_多角形作成機能

デバイスの多角形作成機能を表わします。

表 7.71
多角形作成機能定数

定数	意味
PC_INTERIORS	インテリア作成。
PC_NONE	多角形のサポートなし。
PC_POLYGON	多角形作成。
PC_RECTANGLE	矩形作成。
PC_SCANLINE	走査ライン作成。
PC_STYLED	スタイル境界線作成。
PC_TRAPEZOID	台形作成。
PC_WIDE	太い境界線作成。
PC_WIDESTYLED	太いスタイル境界線作成。
PC_WINDPOLYGON	全体塗りつぶしの多角形作成。

PM_ピークメッセージオプション

PeekMessage 関数によって処理されるメッセージの処理方法を指定します。

表 7.72
PeekMessage オプション

定数	意味
PM_NOREMOVE	処理後のメッセージは除去されずにキューに残ります。
PM_NOYIELD	現処理に対する割り込みをいっさい禁止します。
PM_REMOVE	処理後のメッセージはキューから除去されます。

PolyFill モード

複数の閉鎖領域を持つ多角形を塗りつぶす方法を指定します。***CreatePolygonRgn, CreatePolyPolygonRgn, GetPolyFillMode, SetPolyFillMode*** の各関数の呼び出しで使用します。

表 7.73
PolyFill モード

定数	意味
ALTERNATE	領域を1つおきに塗りつぶします。
WINDING	各領域をすべて塗りつぶします。

PR_スプーラステータスコード

PR_JOBSTATUS は，***WM_SPOOLERSTATUS*** メッセージの *wParam* で渡される値です。

Printer escape（プリンタ拡張）コード

Escape 関数に渡されます。GDI に対応しないデバイス固有の機能に，直接アクセスできるようになります。

表 7.74 プリンタエスケープコード

定数	意味
ABORTDOC	現在の印刷を中止します。
BANDINFO	***Escape*** の呼び出しで渡すレコードに，プリンタのバンディング機能情報を複写します。
BEGIN_PATH	パスをオープンします。パスはプリミティブを連続させた多角形または折れ線を形成します。パスを使うと，PostScript プリンタに送る命令が単純化されます。
CLIP_TO_PATH	現在オープンしているパスにしたがって，クリッピング領域を定義します。
DEVICEDATA	***PASSTHROUGH*** に同じです。
DRAFTMODE	ドラフトモードのオン/オフを行ないます。
DRAWPATTERNRECT	レーザプリンタでグレイスケールまたは黒のパターンの矩形を描きます。
ENABLEDUPLEX	両面印刷機能を使用可能にします。
ENABLEPAIRKERNING	プリンタの文字ペアを文字詰めする機能のオン/オフを行ないます。
ENABLERELATIVEWIDTHS	相対文字幅のオン/オフを行ないます。オフの状態では，文字列の幅が個々の文字幅の合計に等しくなります。オンの状態では，フォントの文字幅テーブルにアクセスして文字列幅の計算を行ないます。
END_PATH	パスをクローズします。***BEGIN_PATH*** を参照してください。
ENDDOC	***STARTDOC*** で開始された印刷を終了します。***STARTDOC*** を参照してください。

表 7.74 プリンタエスケープコード（つづき）

定数	意味
ENUMPAPERBINS	用紙ビン情報を取得します。このエスケープは旧バージョンとの互換性のために使われます。新バージョンには ***GETSETPAPERBINS*** を使用します。
ENUMPAPERMETRICS	サポートされている用紙種別数と，印刷可能な矩形サイズを検索します。このエスケープは，旧バージョンとの互換性のために使われます。新バージョンには ***ExtDeviceMode*** 関数を使います。
EPSPRINTING	PostScript ヘッダ制御セクションを抑制して，GDI 呼び出しを無効にします。
EXT_DEVICE_CAPS	***GetDeviceCaps*** 関数呼び出しでは得られないデバイス固有のプリンタ機能情報を取得します。
EXTTEXTOUT	***ExtTextOut*** 関数の機能に同じです。
FLUSHOUTPUT	プリンタのバッファをクリアします。
GETCOLORTABLE	RGB カラーテーブルエントリを取得します。
GETEXTENDEDTEXTMETRICS	選択されたフォント用の拡張テキストメトリックスを取得します。
GETEXTENTABLE	選択されたフォント文字セットの文字列の幅を取得します。
GETPAIRKERNTABLE	文字ペアカーニングテーブルを取得します。
GETPHYSPAGESIZE	物理ページサイズを取得します。
GETPRINTINGOFFSET	印刷または描画の開始点であるページの左上隅からのオフセットを取得します。
GETSCALINGFACTOR	X 軸および Y 軸の倍率を取得します。
GETSETPAPERBINS	用紙ビン数を取得して，現在の用紙ビンを設定します。
GETSETPAPERMETRICS	用紙メトリックス情報を取得して新情報を設定します。このエスケープは，旧バージョンとの互換性のために使われます。新バージョンには ***ExtDeviceMode*** 関数を使用します。
GETSETPRINTORIENT	用紙の挿入方向の取得または設定を行ないます。このエスケープは，旧バージョンとの互換性のために使われます。新バージョンには ***ExtDeviceMode*** 関数を使用します。
GETTECHNOLOGY	PostScript などの一般的な技術タイプを特定する文字列を取得します。
GETTRACKKERNTABLE	現在の選択フォントのトラックカーニングテーブルを取得します。
GETVECTORBRUSHSIZE	閉鎖形を塗りつぶすペンの幅を，デバイス単位で取得します。

表 7.74
プリンタエスケープコード（つづき）

定数	意味
GETVECTORPENSIZE	ハッチブラシパターンに使用するペンの幅を，デバイス単位で取得します。
MFCOMMENT	メタファイルにコメントを加えます。
NEWFRAME	次ページに進みます。
NEXTBAND	バンドへの出力完了をデバイスドライバに知らせます。
PASSTHROUGH	アプリケーションからプリンタへ直接データを送ります。
QUERYESCSUPPORT	特定のエスケープをサポートするかどうかを判定します。
RESTORE_CTM	保存済みの変換マトリックスを復元します。
SAVE_CTM	現在の変換マトリックスを保存します。
SELECTPAPERSOURCE	現在は更新されて ***GETSETPAPERBINS*** になっています。旧バージョンとの互換性のために残されています。
SET_ARC_DIRECTION	***Arc*** 関数で描く円弧の方向を指定します。
SET_BACKGROUND_COLOR	バックグラウンド色を設定または取得します。
SET_BOUNDS	作成する画像に境界矩形を設定します。
SET_CLIP_BOX	前のクリッピング矩形を設定または復元します。
SET_POLY_MODE	***Polygon***，***PolyLine*** の両関数を呼び出すポリモードを設定します。
SET_SCREEN_ANGLE	色分解を行なうときにフォトグラフィックマスクの傾きをシミュレートするために，現在のスクリーン角度をセットします。
SET_SPREAD	再現プロセスの不完全さを補うのに必要な白以外の原色の拡大量を設定します。
SETABORTPROC	現在の印刷処理を中止する関数を設定します。
SETALLJUSTVALUES	出力するテキストの位置合わせ値を設定します。
SETCOLORTABLE	RGB カラーテーブルエントリを設定します。
SETCOPYCOUNT	印刷部数を指定します。
SETKERNTRACK	使用するカーニングトラックを指定します。
SETLINECAP	線分のエンドキャップを設定します。エンドキャップは四角形や円形でもかまいません。線分の終点を延長したり，フラッシュすることもできます。
SETLINEJOIN	2 線の交差部分をどのようにするかを指定します。丸，直角，斜めの角の 3 種類があります。
SETMITERLIMIT	斜接限界，すなわち斜め継ぎのかわりに，ベベル結合を使うときの角度を設定します。
STARTDOC	デバイスドライバに新たなジョブの開始を知らせます。
STRETCHBLT	***StretchBlt*** 関数の機能に同じです。
TRANSFORM_CTM	現在の変換マトリックスを変更します。

PROC_定数

Module 引数で渡されたモジュールハンドルがデバイスドライバである場合，**GetProcAddress** 関数の *ProcName* 引数として渡されます。

PROC_EXTDEVICEMODE
PROC_DEVICECAPABILITIES
PROC_OLDDEVICEMODE

PS_ペンスタイル

ペンのスタイルを指定します。**CreatePen** 関数で使用します。

表 7.75
ペンスタイル

定数	意味
PS_SOLID	————
PS_DASH	- - - -
PS_DOT	· · · ·
PS_DASHDOT	- · - · - · -
PS_DASHDOTDOT	- ·· - ·· -
PS_NULL	表示なし。
PS_INSIDEFRAME	多角形と折れ線を除くプリミティブの枠の内側の描画に使用。

R2_バイナリラスタ操作定数

すでにある色と現在のペンの色を組み合わせる描画モードを指定します。***GetROP2*** と ***SetROP2*** の両関数の引数として使います。

表 7.76
バイナリラスタ操作定数

定数	意味
R2_BLACK	常に黒。
R2_COPYPEN	常にペンの色。
R2_MASKNOTPEN	すでにある表示色と現在のペンの反転色に共通な色の組み合わせ。
R2_MASKPEN	すでにある表示色と現在のペンの両方に共通な色の組み合わせ。
R2_MASKPENNOT	***R2_MASKPEN*** の色の反転色。
R2_MERGENOTPEN	現在のペン色の反転色とすでにある表示色の組み合わせ。
R2_MERGEPEN	現在のペン色とすでにある表示色の組み合わせ。
R2_MERGEPENNOT	現在のペン色とすでにある表示色の反転色の組み合わせ。
R2_NOP	すでにある表示色に変更なし。
R2_NOT	すでにある表示色の反転色。
R2_NOTCOPYPEN	現在のペン色の反転色。
R2_NOTMASKPEN	***R2_MASKPEN*** モードによる組み合わせ色の反転色。
R2_NOTMERGEPEN	***R2_MERGEPEN*** モードによる組み合わせ色の反転色。
R2_NOTXORPEN	***R2_XORPEN*** モードによる色の反転色。
R2_WHITE	白色。
R2_XORPEN	すでにある表示色と現在のペン色の組み合わせ色。ただしすでにある表示色と現在のペン色の両方に含まれる色の組み合わせはなし。

RC_ラスタ機能

デバイスのラスタ機能を表わします。

表 7.77
ラスタ機能定数

定数	意味
RC_BANDING	バンディングのサポートを要求。
RC_BIGFONT	64 K を超えるフォントをサポート。
RC_BITBLT	ビットマップの転送可能。
RC_BITMAP64	64 K を超えるビットマップをサポート。
RC_DI_BITMAP	メモリの DIB をサポート。
RC_DIBTODEV	***DIBitsToDevice*** 関数をサポート。
RC_FLOODFILL	***FloodFill*** 関数をサポート。
RC_GDI20_OUTPUT	Windows 2.0 機能をサポート。
RC_PALETTE	パレットをサポート。
RC_SCALING	スケーリング機能のサポートを要求。
RC_STRETCHBLT	***StretchBlt*** 関数をサポート。
RC_STRETCHDIB	***StretchDIBits*** 関数をサポート。

Region（領域）フラグ

グラフィックス領域のタイプを指定します。***CombineRgn, ExcludeClipRect, ExcludeUpdateRgn, GetClipBox, GetRgnBox, GetUpdateRgn, OffsetClipRgn, OffsetRgn, SelectClipRgn*** の各関数の戻り値として使います。

表 7.78
グラフィックス領域フラグ

定数	意味
COMPLEXREGION	領域内で重なり合う境界あり。
ERROR	領域の作成なし。
NULLREGION	空領域。
SIMPLEREGION	領域内で重なり合う境界なし。

RGN_領域結合フラグ

CombineRgn 関数で使う 2 領域の結合方法を指定します。

表 7.79
領域結合フラグ

定数	意味
RGN_AND	2 領域が重なり合う部分。
RGN_COPY	1 つめの領域のコピー。
RGN_DIFF	1 つめの領域の中で 2 つめの領域と重なり合わない部分。
RGN_OR	2 領域のどちらかに含まれる部分。
RGN_XOR	2 領域の重なり合わない部分。

RT_リソース型

Windows リソースの使用可能な型を表わします。***FindResource*** 関数の引数として使います。

表 7.80
リソース型定数

定数	意味
RT_ACCELERATOR	アクセラレータテーブル。
RT_BITMAP	ビットマップ。
RT_CURSOR	カーソル。
RT_DIALOG	ダイアログボックステンプレート。
RT_FONT	フォント。
RT_FONTDIR	フォントディレクトリ。
RT_ICON	アイコン。
RT_MENU	メニュー。
RT_RCDATA	ユーザ定義リソース（データ）。
RT_STRING	文字列。

S_サウンド定数

さまざまなサウンド関数に使用される定数で，使用される関数によって分類されます。

エラー値

下記のエラー値がサウンド関数から返されます。

表 7.81
サウンド定数

定数	意味
S_SERBDNT	無効符点。
S_SERDCC	無効符点数。
S_SERDDR	無効持続時間。
S_SERDFQ	無効周波数。
S_SERDLN	無効音長。
S_SERDMD	無効モード。
S_SERDPT	無効ピッチ。
S_SERDSH	無効波形。
S_SERDSR	無効ソース。
S_SERDST	無効状態。
S_SERDTP	無効テンポ。
S_SERDVL	無効ボリューム。
S_SERDVNA	デバイス使用不可。
S_SERMACT	音楽が演奏中。
S_SEROFM	メモリ不足。
S_SERQFUL	キューがいっぱい。

SetSoundNoise

ノイズソースを指定します。*N* は目標周波数を導き出す値です。

表 7.82
ノイズソース定数

定数	意味
S_PERIOD512	ハイピッチ，N/512　弱ヒス音。
S_PERIOD1024	中ピッチ，N/1024。
S_PERIOD2048	ローピッチ，N/2048　強ヒス音。
S_PERIODVOICE	音声チャンネル3。
S_WHITE512	ハイピッチ，N/512　弱ヒス音。
S_WHITE1024	中ピッチ，N/1024。
S_WHITE2048	ローピッチ，N/2048　強ヒス音。
S_WHITEVOICE	音声チャンネル3。

SetVoiceAccent

演奏される音符のアクセントを指定します。

表 7.83
音符のアクセントオプション

定数	意味
S_LEGATO	音符を時間いっぱいに延ばして，次の音符の始まりと重ねます。
S_NORMAL	音符を時間いっぱいに延ばして，次の音符が始まる前に止めます。
S_STACCATO	音符はその時間より短く演奏し，次の音符との間にはっきりとポーズを入れます。

WaitSoundState

音声キューの状態を指定します。

表 7.84
音声キューオプション

定数	意味
S_ALLTHRESHOLD	音声キューがすべて限界値に到達。
S_QUEUEEMPTY	音声キューがすべて空で，サウンドドライバがオフ状態。
S_THRESHOLD	音声キューの1つが限界値に到達して音声を返却。

SB_スクロールバーコマンド

スクロールバーの動作を指定します。*WM_HSCROLL*, *WM_HSCROLLCLIPBOARD*, *WM_VSCROLL*, *WM_VSCROLLCLIPBOARD* の各メッセージの引数として渡されます。

表 7.85
スクロールバー動作定数

定数	意味
SB_BOTTOM	下方または右方へのスクロール。
SB_ENDSCROLL	スクロール終了。
SB_LINEDOWN	1 行下へのスクロール。
SB_LINEUP	1 行上へのスクロール。
SB_PAGEDOWN	次ページへのスクロール。
SB_PAGEUP	前ページへのスクロール。
SB_THUMBPOSITION	スクロールボックスの絶対位置へのドラッグ。
SB_THUMBTRACK	スクロールボックスの指定位置へのドラッグ。
SB_TOP	上方または左方へのスクロール。

SB_スクロールバー定数

GetScrollPos, *GetScrollRange*, *SetScrollPos*, *SetScrollRange*, *ShowScrollBar* の各関数に指定される特定のスクロールバーを指定します。

S
Cons

表 7.86
スクロールバー定数

定数	意味
SB_BOTH	ウィンドウの水平，垂直のスクロールバーを指定します。*SB_BOTH* の使用は *ShowScrollBar* 関数に限ります。
SB_CTL	スタンドアロンのスクロールバー制御を指定します。
SB_HORZ	ウィンドウの水平スクロールバーを指定します。
SB_VERT	ウィンドウの垂直スクロールバーを指定します。

SBS_スクロールバースタイル

CreateWindow，***CreateWindowEx*** の両関数によって作成するスクロールバーのスタイルを指定します。

表 7.87
スクロールバースタイル

定数	**意味**
SBS_BOTTOMALIGN	高さがデフォルトのスクロールバーです。その下端は，スクロールバー作成矩形の下端に揃えられます。***SBS_BOTTOMALIGN*** スタイルの使用は，***SBS_HORZ*** スタイルと併用する場合に限られます。
SBS_HORZ	水平スクロールバーです。***SBS_BOTTOMALIGN*** スタイルも ***SBS_TOPALIGN*** スタイルも指定されていない場合，バーのサイズは作成時に指定されたままです。
SBS_LEFTALIGN	幅がデフォルトのスクロールバーです。その左端は，スクロールバー作成矩形の左端に揃えられます。***SBS_LEFTALIGN*** スタイルの使用は，***SBS_VERT*** スタイルと併用する場合に限られます。
SBS_RIGHTALIGN	幅がデフォルトのスクロールバーです。その右端は，スクロールバー作成矩形の右端に揃えられます。***SBS_RIGHTALIGN*** スタイルの使用は，***SBS_VERT*** スタイルと併用する場合に限られます。
SBS_SIZEBOX	サイズボックスです。***SBS_SIZEBOXBOTTOMRIGHTALIGN*** スタイルも ***SBS_SIZEBOXTOPLEFTALIGN*** スタイルも指定されていない場合，スクロールバーのサイズは作成時に指定されたままです。
SBS_SIZEBOXBOTTOMRIGHTALIGN	サイズボックスがデフォルトのスクロールバーです。その右下隅は，スクロールバー作成矩形の右下隅に揃えられます。***SBS_SIZEBOXBOTTOMRIGHTALIGN*** スタイルの使用は，***SBS_SIZEBOX*** スタイルと併用する場合に限ります。
SBS_SIZEBOXTOPLEFTALIGN	サイズボックスがデフォルトのスクロールバーです。その左上隅は，スクロールバー作成矩形の左上隅に揃えられます。***SBS_SIZEBOXTOPLEFTALIGN*** スタイルの使用は，***SBS_SIZEBOX*** スタイルと併用する場合に限ります。

表 7.87
スクロールバースタイル（つづき）

定数	意味
SBS_TOPALIGN	高さがデフォルトのスクロールバーです。その上端は，スクロールバー作成矩形の上端に揃えられます。***SBS_TOPALIGN*** スタイルの使用は，***SBS_HORZ*** スタイルと併用する場合に限ります。
SBS_VERT	垂直スクロールバーです。***SBS_RIGHTALIGN*** スタイルも ***SBS_LEFTALIGN*** スタイルも指定されていない場合はバーのサイズは作成時に指定されたままです。

SC_システムコマンド値

最小化ボタンか最大化ボタンのクリック，またはコントロールメニューの項目選択に応じて，***WM_SYSCOMMAND*** メッセージに渡されるコマンドです。要求された処理を表わします。

表 7.88
システムコマンド定数

定数	意味
SC_CLOSE	ウィンドウのクローズ。
SC_HSCROLL	水平スクロール。
SC_KEYMENU	キーストロークによるメニュー取得。
SC_MAXIMIZE	ウィンドウの最大化。
SC_MINIMIZE	ウィンドウの最小化。
SC_MOUSEMENU	マウスのクリックによるメニュー取得。
SC_MOVE	ウィンドウの移動。
SC_NEXTWINDOW	次のウィンドウに移動。
SC_PREVWINDOW	前のウィンドウに移動。
SC_RESTORE	最大化または最小化状態のウィンドウを復元。
SC_SIZE	ウィンドウのサイズ変更。
SC_TASKLIST	タスクリストの表示。
SC_VSCROLL	垂直スクロール。

SHOW_ShowWindow 旧コマンド

Windowsの旧バージョン用との互換性のために残されている旧い定数です。現バージョンでは，*SW_*ウィンドウ表示定数を使用します。

表7.89
ShowWindow旧コマンド

定数	値	意味
HIDE_WINDOW	0	ウィンドウ非表示。
SHOW_OPENWINDOW	1	アイコンからウィンドウを復元。
SHOW_ICONWINDOW	2	ウィンドウをアイコン化。
SHOW_FULLSCREEN	3	ウィンドウを画面全体に拡大。
SHOW_OPENNOACTIVATE	4	ウィンドウの復元，ただし非アクティブのまま。

Size（サイズ）変更定数

ウィンドウのサイズ変更を示します。*WM_SIZE*メッセージで渡されます。

表7.90
ウィンドウサイズ変更定数

定数	意味
SIZEFULLSCREEN	最大化。
SIZEICONIC	最小化。
SIZENORMAL	最大または最小以外のサイズに変更。
SIZEZOOMHIDE	他ウィンドウの最大化。
SIZEZOOMSHOW	最大化状態の他ウィンドウを復元。

SM_システムメトリックスコード

*GetSystemMetrics*関数によって寸法を取得するWindowsユーザインターフェースの各アスペクトを指定します。

表 7.91
システムメトリックスコード

定数	意味
SM_CXBORDER	変更不能なウィンドウフレーム幅。
SM_CXCURSOR	カーソル幅。
SM_CXDLGFRAME	*WM_DLGFRAME* スタイルのウィンドウフレーム幅。
SM_CXFRAME	変更可能なウィンドウフレーム幅。
SM_CXFULLSCREEN	最大サイズのウィンドウのクライアント領域幅。
SM_CXHSCROLL	水平スクロールバーの矢印幅。
SM_CXHTHUMB	水平スクロールボックスの幅。
SM_CXICON	アイコン幅。
SM_CXMIN	ウィンドウの最小幅。
SM_CXMINTRACK	ウィンドウの最小トラッキング幅。
SM_CXSCREEN	画面幅。
SM_CXSIZE	タイトルバーのビットマップ幅。
SM_CXVSCROLL	垂直スクロールバーの矢印幅。
SM_CYBORDER	変更不能なウィンドウフレームの高さ。
SM_CYCAPTION	キャプションの高さ。
SM_CYCURSOR	カーソルの高さ。
SM_CYDLGFRAME	*WS_DLGFRAME* スタイルのウィンドウフレームの高さ。
SM_CYFRAME	変更可能なウィンドウフレームの高さ。
SM_CYFULLSCREEN	ウィンドウ最大化時のクライアント領域の高さ。
SM_CYHSCROLL	水平スクロールバー矢印の高さ。
SM_CYICON	アイコンの高さ。
SM_CYKANJIWINDOW	漢字ウィンドウの高さ。
SM_CYMENU	単一行メニューの高さ。
SM_CYMIN	ウィンドウの高さの最小値。
SM_CYMINTRACK	ウィンドウのトラッキングの高さの最小値。
SM_CYSCREEN	画面の高さ。
SM_CYSIZE	タイトルバーのビットマップの高さ。
SM_CYVSCROLL	垂直スクロールバーの矢印の高さ。
SM_CYVTHUMB	垂直スクロールボックス幅。
SM_DEBUG	デバッグ用バージョン以外の Windows の場合は 0 を返します。
SM_MOUSEPRESENT	マウスが存在しない場合は 0 を返します。
SM_SWAPBUTTON	マウスボタンがスワップされていなければ 0 を返します。

SP_スプーラエラーコード

Escape 関数からエラーコードとして返される定数です。

表 7.92
スプーラエラーコード

定数	意味
SP_APPABORT	アプリケーションによるジョブ中断。
SP_ERROR	通常エラー。
SP_OUTOFDISK	ディスク空き容量不足でスプーリング不可。
SP_OUTOFMEMORY	メモリ不足でスプーリング不可。
SP_USERABORT	ユーザによるスプーラのジョブ中断。

SS_スタティックコントロールスタイル

CreateWindow, ***CreateWindowEx*** の両関数で作成するスタティックコントロールのスタイルを指定します。

表 7.93
スタティックコントロールスタイル

定数	意味
SS_BLACKFRAME	ウィンドウのフレームと同色のフレームになります。
SS_BLACKRECT	ウィンドウフレームの色で塗りつぶされます。
SS_CENTER	矩形の中央にテキストを表示します。コントロール幅を超えるテキスト行はラップされて改行されます。改行は単語間で行なわれ，各行は新たにセンタリングされます。
SS_GRAYFRAME	画面のバックグラウンド色と同色のフレームになります。
SS_GRAYRECT	画面のバックグラウンド色で塗りつぶされます。
SS_ICON	アイコンです。コントロールのテキストは，リソースファイルの中のアイコン名です。アイコンは自動的に自分のサイズを決めます。
SS_LEFT	矩形に左詰めでテキストを表示します。コントロール幅を超えるテキスト行はラップされて改行されます。改行は単語間で行なわれ，各行は左に揃えられます。
SS_LEFTNOWORDWRAP	矩形に左詰めでテキストを表示します。コントロール幅を超えるテキストはクリップされます。

表 7.93
スタティックコントロールスタイル（つづき）

定数	意味
SS_NOPREFIX	テキスト内の‘&’文字を無視します。‘&’はアクセラレータのプレフィックス文字として使用されることが多く，これを文字列から除去して次の文字に下線をつけます。
SS_RIGHT	矩形に右詰めでテキストを表示します。コントロール幅を超えるテキスト行はラップされて改行されます。改行は単語間で行なわれ，各行は右に揃えられます。
SS_SIMPLE	左端から1行のテキストを表示します。テキストの位置変更はできません。コントロールの親ウィンドウは ***WM_CTLCOLOR*** メッセージを処理できません。
SS_USERITEM	ユーザ定義のスタティックコントロールです。
SS_WHITEFRAME	ウィンドウのバックグラウンド色と同色のフレームになります。
SS_WHITERECT	ウィンドウのバックグラウンド色で塗りつぶされます。

Stock logical object（ストック論理オブジェクト）

定義済みのストック描画ツールを表わします。***GetStockObject*** 関数で使用します。

表 7.94
ストック論理オブジェクト定数

定数	意味
BLACK_BRUSH	黒ブラシ。
DKGRAY_BRUSH	ダークグレイブラシ。
GRAY_BRUSH	グレイブラシ。
HOLLOW_BRUSH	中空ブラシ。
LTGRAY_BRUSH	ライトグレイブラシ。
NULL_BRUSH	ヌルブラシ。
WHITE_BRUSH	白ブラシ。
BLACK_PEN	黒ペン。
NULL_PEN	ヌルペン。
WHITE_PEN	白ペン。
ANSI_FIXED_FONT	ANSI 固定ピッチシステムフォント。
ANSI_VAR_FONT	ANSI 可変ピッチシステムフォント。

表 7.94
ストック論理オブジェクト定数（つづき）

定数	意味
DEVICE_DEFAULT_FONT	デバイス依存フォント。
OEM_FIXED_FONT	OEM 依存固定フォント。
SYSTEM_FIXED_FONT	旧バージョン Windows の固定ピッチフォント。
SYSTEM_FONT	可変ピッチの Windows システムフォント。
DEFAULT_PALETTE	デフォルトの色パレット。

StretchBlt モード

GetStretchBltMode，***SetStretchBltMode*** の両関数で使用するビットマップストレッチモードです。

表 7.95
ビットマップストレッチモード

定数	意味
BLACKONWHITE	AND（&）演算子で，ビットマップの黒部分を残して行を消去します。
COLORONCOLOR	内容にかかわらず行をすべて消去します。行の情報は処理中に失なわれます。
WHITEONBLACK	OR（|）演算子で，ビットマップの白部分を残して行を消去します。

SW_ウィンドウ表示定数

ShowWindow 関数によるウィンドウの表示形態を示します。

表 7.96
ShowWindow 定数

定数	意味
SW_HIDE	非表示。
SW_MAXIMIZE	***SW_SHOWMAXIMIZED*** に同じ。
SW_MINIMIZE	最小化および非アクティブ化。

表 7.96
ShowWindow 定数
(つづき)

定数	意味
SW_NORMAL	*SW_SHOWNORMAL* に同じ。
SW_OTHERZOOM	別ウィンドウを最大化，ただし 2.0 バージョン互換用。
SW_OTHERUNZOOM	別ウィンドウを最小化，ただし 2.0 バージョン互換用。
SW_RESTORE	*SW_SHOWNORMAL* に同じ。
SW_SHOW	現在のサイズと位置で表示。
SW_SHOWMAXIMIZED	最大化およびアクティブ化。
SW_SHOWMINIMIZED	最小化およびアクティブ化。
SW_SHOWMINNOACTIVE	アクティブなままで最小化。
SW_SHOWNA	アクティブなままで現状態表示。
SW_SHOWNOACTIVATE	アクティブなままで現在のサイズと位置表示。
SW_SHOWNORMAL	復元およびアクティブ化。

SW_ウィンドウメッセージ表示定数

ウィンドウの表示状態を示す識別子です。***WM_SHOWWINDOW*** メッセージで渡されます。

表 7.97
WM_SHOWWINDOW
定数

定数	意味
SW_PARENTCLOSING	親ウィンドウの最小化またはポップアップウィンドウの非表示。
SW_PARENTOPENING	親ウィンドウまたはポップアップウィンドウの表示。

SWP_ウィンドウ位置設定フラグ

SetWindowPos, DeferWindowPos の両関数で渡されます。指定ウィンドウに対する処理を示します。

表 7.98
ウィンドウ位置設定フラグ

定数	意味
SWP_DRAWFRAME	ウィンドウクラスが定義するウィンドウフレームを描画。
SWP_HIDEWINDOW	ウィンドウを非表示。
SWP_NOACTIVATE	ウィンドウはアクティブ化しない。
SWP_NOMOVE	ウィンドウ位置を入力された座標位置に移動しない。
SWP_NOREDRAW	変更があってもウィンドウの再表示なし。
SWP_NOSIZE	ウィンドウを入力された縦横寸法に変更しない。
SWP_NOZORDER	ウィンドウの順序に変更なし。
SWP_SHOWWINDOW	ウィンドウの表示。

SYSPAL_システムパレットフラグ

システムパレットの使用を指定します。***GetSystemPaletteUse, SetSystemPaltteUse*** の両関数で使用されます。

表 7.99
システムパレットフラグ

定数	意味
SYSPAL_NOSTATIC	黒と白以外のスタティックカラーなし。
SYSPAL_STATIC	アプリケーションによって実現される論理パレットに影響されないスタティックカラー。

TA_テキスト配置オプション

TextOut, ***ExtTextOut*** の両関数によって描かれるテキストの配置を制御します。***GetTextAlign***, ***SetTextAlign*** の両関数で指定，または返されるオプションです。

表 7.100
テキスト配置オプション

定数	意味
TA_BASELINE	現在のフォントのベースラインに配置。
TA_BOTTOM	矩形の下辺に配置。
TA_CENTER	矩形の横中央に配置。
TA_LEFT	矩形の左辺に配置（デフォルト）。
TA_NOUPDATECP	***TextOut*** または ***ExtTextOut*** の呼び出しに伴う位置更新なし（デフォルト）。
TA_RIGHT	矩形の右辺に配置。
TA_TOP	矩形の上辺に配置（デフォルト）。
TA_UPDATECP	***TextOut*** または ***ExtTextOut*** の呼び出しに伴い位置更新。

TC_テキスト機能

デバイスのテキスト作成機能を表わします。

表 7.101
テキスト機能定数

定数	意味
TC_CP_STROKE	ストロームクリップ精度。
TC_CR_90	90° の文字回転。
TC_CR_ANY	あらゆる角度の文字回転。
TC_EA_DOUBLE	ダブルウェイト文字。
TC_IA_ABLE	イタリック文字。
TC_OP_CHARACTER	文字出力精度。
TC_OP_STROKE	ストローク出力精度。
TC_RA_ABLE	ラスタフォント。

表 7.101
テキスト機能定数（つづき）

定数	意味
TC_SA_CONTIN	あらゆる倍率の計算。
TC_SA_DOUBLE	倍角文字。
TC_SA_INTEGER	整数倍の倍率計算。
TC_SF_X_YINDEP	x 座標と y 座標に依存しない拡大縮小。
TC_SO_ABLE	ストライクアウトつき文字。
TC_UA_ABLE	下線つき文字。
TC_VA_ABLE	ベクトルフォント。

TF_FORCEDRIVE GetTempFileName フラグ

GetTempFileName 関数呼び出しで *DriveLetter* 引数と ***TF_FORCEDRIVE*** を組み合わせると，指定ドライブに一時ファイルが作成されます。その他の場合は，最初のハードディスクまたは *temp* 環境変数で指定されたディレクトリに作成されます。

Ternary raster operation（3 次元ラスタ操作）

BitBlt, PatBlt, StretchBlt, StretchDIBits の各関数のラスタ操作コードとして使用されます。

表 7.102
3 次元ラスタ操作定数

定数	意味
BLACKNESS	黒一色の出力。
DSTINVERT	反転ビットマップ。
MERGECOPY	論理演算 AND（&）によってパターンと転送元ビットマップを結合。
MERGEPAINT	論理演算 OR（|）によって転送先のビットマップと反転した転送元ビットマップを結合。
NOTSRCCOPY	転送元ビットマップを反転して，転送先ビットマップに複写。
NOTSRCERASE	***MERGEPAINT*** の結果を反転。

表 7.102
3次元ラスタ操作定数（つづき）

定数	意味
PATCOPY	ビットマップに指定パターンを複写。
PATINVERT	論理演算 XOR（^）によって転送先ビットマップとパターンを結合。
PATPAINT	論理演算 OR（|）によって転送元ビットマップとパターンを結合後，さらに OR によって転送先ビットマップと結合。
SRCAND	論理演算 AND（&）によって転送元と転送先ビットマップを結合。
SRCCOPY	転送元ビットマップを転送先に複写。
SRCERASE	論理演算 AND（&）によって転送元と反転した転送先を結合。
SRCINVERT	論理演算 XOR（^）によって転送元と転送先ビットマップを結合。
SRCPAINT	論理演算 OR（|）によって転送元と転送先ビットマップを結合。
WHITENESS	白一色の出力。

VK_仮想キーコード

文字キーの ***A*** や ***F1*** キーなどの標準キーを表わす定数です。メーカーによって配列と機能の異なるキーボードからの入力を統一します。この仮想キーコードを使用して，アクセラレータキーの定義，または ***WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN, WM_SYSKEYUP*** などのキーボードメッセージの処理を行ないます。下表は仮想キーコードと対応キーの一覧です。

表 7.103
標準仮想キーセット

コード	キーまたはボタン
VK_LBUTTON	左マウスボタン。
VK_RBUTTON	右マウスボタン。
VK_CANCEL	***STOP***
VK_MBUTTON	中央マウスボタン。
VK_BACK	***BS***
VK_TAB	***TAB***
VK_CLEAR	***HOME/CLR***

表 7.103
標準仮想キーセット（つづき）

コード	キーまたはボタン
VK_RETURN	リターン
VK_SHIFT	***SHIFT***
VK_CONTROL	***CTRL***
VK_MENU	***GRPH***
VK_PAUSE	ポーズ
VK_CAPITAL	***CAPS***
VK_KANA	カナキー
VK_ESCAPE	***ESC***
VK_SPACE	スペース
VK_PRIOR	***ROLLDOWN***
VK_NEXT	***ROLLUP***
VK_END	***SHIFT-HOME***
VK_HOME	***HOME*** /***CLR***
VK_LEFT	左矢印
VK_UP	上矢印
VK_RIGHT	右矢印
VK_DOWN	下矢印
VK_SELECT	選択
VK_PRINT	OEM 固有
VK_EXECUTE	実行
VK_SNAPSHOT	***COPY***
VK_INSERT	***INS***
VK_DELETE	***DEL***
VK_HELP	***HELP***
VK_A TO VK_Z	***A*** から ***Z*** のキー
VK_0 TO VK_9	***0*** から ***9*** のキー
VK_NUMPAD 0 ～ *VK_NUMPAD 9*	テンキーの ***0*** から ***9***
VK_MULTIPLY	乗算（テンキーの‘*’）
VK_ADD	加算（テンキーの‘+’）
VK_SEPARATOR	分離記号（テンキーの‘,’）
VK_SUBTRACT	減算（テンキーの‘-’）
VK_DECIMAL	小数点（テンキーの‘.’）
VK_DIVIDE	除算（テンキーの‘/’）
VK_F1 ～ *VK_F16*	***F1*** から ***F16*** のファンクションキー
VK_NUMLOCK	***Num Lock***
VK_OEM_SCROLL	***CTR*** ＋テンキーの‘=’

表 7.103
標準仮想キーセット（つづき）

コード	キーまたはボタン
VK_OEM_EQU	テンキーの '='
VK_OEM_1	コロン（':' キー）
VK_OEM_PLUS	セミコロン（';' キー）
VK_OEM_COMMA	カンマ（',' キー）
VK_OEM_MINUS	マイナス（'−' キー）
VK_OEM_PERIOD	ピリオド（'.' キー）
VK_OEM_2	スラッシュ（'/' キー）
VK_OEM_3	アットマーク（'@' キー）
VK_OEM_4	左角カッコ（'[' キー）
VK_OEM_5	円記号（'¥' キー）
VK_OEM_6	右角カッコ（']' キー）
VK_OEM_7	キャレット（'^' キー）
VK_OEM_8	カナ＋ 'ロ' キー

WEP_DLL 終了コード

ダイナミックリンクライブラリからアクセス可能な ***ExitCode*** 変数に使います。

表 7.104
DLL 終了コード

定数	意味
WEP_FREE_DLL	DLL のみの終了とメモリからの除去。
WEP_SYSTEM_EXIT	Windows 自体の終了。

WF_Windows メモリコンフィギュレーションフラグ

Windowsのメモリコンフィギュレーションを指定します。***GetWinFlags*** 関数の戻り値として使います。

表 7.105
Windows メモリコンフィギュレーションフラグ

定数	意味
WF_80X87	数値演算コプロセッサ搭載。
WF_CPU086	CPU 8086 搭載。
WF_CPU186	CPU 80186 搭載。
WF_CPU286	CPU 80286 搭載。
WF_CPU386	CPU 80386 搭載。
WF_CPU486	CPU 80486 搭載。
WF_ENHANCED	386 エンハンス（プロテクト）モードで動作する Windows。
WF_LARGEFRAME	ラージフレームコンフィギュレーションのシステム。
WF_PMODE	386 エンハンスまたはスタンダード（プロテクト）モードで動作する Windows。
WF_SMALLFRAME	スモールフレームコンフィギュレーションのシステム。
WF_STANDARD	スタンダード（プロテクト）モードで動作する Windows。
WF_WIN286	***WF_STANDARD*** に同じ，またはスタンダードモード。
WF_WIN386	***WF_ENHANCED*** に同じ，または 386 エンハンスモード。

WH_Windows フックコード

フィルタ関数チェインにインストールまたはチェインから除去する特定タイプのフィルタ関数を指定します。インストールには ***SetWindowsHook*** 関数，除去には ***UnhookWindowsHook*** 関数を使用します。

表 7.106
Windows フックコード

定数	意味
WH_CALLWNDPROC	ウィンドウ関数フィルタ。
WH_GETMESSAGE	メッセージフィルタ。
WH_JOURNALPLAYBACK	ジャーナルプレイバックフィルタ。

表 7.106
Windows フックコード
(つづき)

定数	意味
WH_JOURNALRECORD	ジャーナルレコードフィルタ。
WH_KEYBOARD	キーボードフィルタ。
WH_MSGFILTER	メッセージフィルタ（***SetWindowsHook*** のみ）。
WH_SYSMSGFILTER	システムワイドメッセージフィルタ（***SetWindowsHook*** のみ）。

WS_ウィンドウスタイル

CreateWindow, ***CreateWindowEx*** の両関数で作成するウィンドウ，ダイアログ，コントロールのウィンドウスタイルを指定します。組み合わせて使う定数です。

表 7.107
ウィンドウスタイル

定数	意味
WS_BORDER	境界線つきのウィンドウです。***WS_DLGFRAME*** スタイルとは併用できません。
WS_CAPTION	タイトルバーと境界線つきのウィンドウです。***WS_DLGFRAME*** スタイルとは併用できません。このスタイルが指定されると，同時に ***WS_BORDER*** も暗黙に指定されます。
WS_CHILD	子ウィンドウです。***WS_POPUP*** スタイルとは併用できません。
WS_CHILDWINDOW	***WS_CHILD*** に同じです。
WS_CLIPCHILDREN	子ウィンドウで覆われる領域以外を描画します。
WS_CLIPSIBLINGS	兄弟ウィンドウをすべてクリップして描画します。親が共通の子ウィンドウのクライント領域では，描画可能領域はオーバーラップしません。このスタイルは必ず ***WS_CHILD*** と併用します。
WS_DISABLED	初期状態が使用不可のウィンドウです。
WS_DLGFRAME	二重境界線つきでタイトルなしのウィンドウです。***WS_BORDER*** スタイルとは併用できません。
WS_GROUP	矢印キーでアクセスできるコントロールグループの最初のコントロールです。このスタイルで定義されないコントロールは，最後に ***WS_GROUP*** スタイルで定義されたコントロールによって起動されるグループに属します。

表 7.107
ウィンドウスタイル（つづき）

定数	意味
WS_HSCROLL	水平スクロールバーつきのウィンドウです。
WS_ICONIC	*WS_MINIMIZE* に同じです。
WS_MAXIMIZE	画面の最大サイズで表示されるウィンドウです。
WS_MAXIMIZEBOX	最大化ボタンつきのウィンドウです。
WS_MINIMIZE	*WS_OVERLAPPED* スタイルとの併用に限ります。最小サイズであるアイコン状態で初期表示されるウィンドウです。
WS_MINIMIZEBOX	最小化ボタンつきのウィンドウです。
WS_OVERLAPPED	オーバーラップウィンドウです。キャプションと境界線つきです。
WS_OVERLAPPEDWINDOW	*WS_OVERLAPPED*, *WS_CAPTION*, *WS_SYSMENU*, *WS_THICKFRAME*, *WS_MINIMIZEBOX*, *WS_MAXIMIZEBOX* をすべて組み合わせたのと同じスタイルになります。
WS_POPUP	ポップアップウィンドウです。*WS_CHILD* スタイルとは併用できません。
WS_POPUPWINDOW	*WS_BORDER*, *WS_POPUP*, *WS_SYSMENU* の各スタイルを組み合わせたのと同じスタイルになります。コントロールメニューボックスを可視状態にするには，*WS_CAPTION* スタイルと併用します。
WS_SIZEBOX	*WS_THICKFRAME* に同じです。
WS_SYSMENU	タイトルバーにコントロールメニューボックスのあるウィンドウです。タイトルバーつきのウィンドウにのみ使用できます。
WS_TABSTOP	**TAB** キーによって移動できるコントロールリストの中のコントロールです。コントロールはダイアログボックスの子に限ります。
WS_THICKFRAME	ウィンドウのサイズ変更用の太い枠つきのウィンドウです。
WS_TILED	*WS_OVERLAPPED* に同じです。
WS_TILEDWINDOW	*WS_OVERLAPPEDWINDOW* に同じです。
WS_VISIBLE	初期状態で表示されるウィンドウです。
WS_VSCROLL	垂直スクロールバーつきのウィンドウです。

WS_EX_拡張ウィンドウスタイル

*WS_*定数と組み合わせて，***CreateWindowEx*** 関数で作成するウィンドウの拡張ウィンドウスタイルを指定します。

表 7.108
拡張ウィンドウスタイル

定数	意味
WS_EX_DLGMODALFRAME	二重境界線つきウィンドウです。*WS_CAPTION* と併用できます。
WS_EX_NOPARENTNOTIFY	子ウィンドウが作成または破棄されても，親ウィンドウに *WM_PARENTNOTIFY* メッセージが送られません。

W
Con

索　引

【A】

【B】

【C】

【D】

【E】

【F】

【G】

【H】

【I】

【K】

【L】

【M】

【N】

【O】

【P】

【R】

【S】

【T】

【U】

【V】

【W】

【い】

【う】

【え】

【お】

【か】

【き】

【く】

【け】

【こ】

【さ】

【し】

【す】

【せ】

【そ】

【た】

【ち】

【つ】

【て】

【と】

【な】

【に】

【ぬ】

【は】

【ひ】

【ふ】

【へ】

【ほ】

【ま】

【め】

【も】

【れ】

【ろ】

ObjectWindows™ for C++　リファレンスガイド

1992年　1月　6日　初版発行

著者　**Borland International**

編訳・発行　**ボーランド株式会社**

〒107 東京都港区南青山7-8-1　小田急南青山ビル

TEL　(03)3486-1400(代表)

FAX　(03)3486-1423

Printed in Japan

落丁，乱丁はお取り替えいたします。


BORLAND® C++
BORLAND
Copyright ©1987, 1991 Borland International, Inc. All rights reserved. Licensed material—program property of Borland International, Inc. 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001. Unauthorized use, duplication or distribution is strictly prohibited by federal law.
APPLICATION FRAMEWORKS
BORLAND® C++
BORLAND
Copyright ©1987, 1991 Borland International, Inc. All rights reserved. Licensed material—program property of Borland International, Inc. 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001. Unauthorized use, duplication or distribution is strictly prohibited by federal law.
APPLICATION FRAMEWORKS
BORLAND® C++
BORLAND
Copyright ©1987, 1991 Borland International, Inc. All rights reserved. Licensed material—program property of Borland International, Inc. 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001. Unauthorized use, duplication or distribution is strictly prohibited by federal law.
APPLICATION FRAMEWORKS
BORLAND® C++
BORLAND

REFERENCE GUIDE

OBJECTWINDOWS™ FOR C++

BORLAND